TURING 图灵程序设计丛书

[PACKT] PUBLISHING



Linux Shell Scripting Cookbook Second Edition

细致剖析实际应用中的110多个案例，让看似复杂的Linux shell脚本任务迎刃而解。

Linux Shell 脚本攻略（第2版）

[印] Shantanu Tushar
Sarath Lakshman 著

门佳 译

人民邮电出版社
POSTS & TELECOM PRESS

数字版权声明

Shantanu Tushar

资深GNU/Linux用户，KDE社区著名的贡献者，维护着Calligra Active（用于Tablets的KDE办公文档查看器）、Plasma Media Center以及Gluon Player。Shantanu坚信终有一天编程会变得无比轻松，每个人都会热衷于为计算机编写程序。

Sarath Lakshman

年轻的Linux天才程序员、开源软件及GNU/Linux活跃分子。他作为SLYNUX(2005)的开发者而广为人知，这是一款面向Linux新手的操作友好的GNU/Linux发布版。另外，他还为Linux For You月刊撰写文章，并且还在Fedora、Pardus Linux、PiTiVi、Ubuntu 以及Google编程夏令营等项目中，他都作出了显著的贡献。

门佳

Unix / Linux shell、Perl、正则表达式爱好者。在2001年接触Linux后很快喜欢上该系统。对Unix / Linux系统管理、Linux内核、Web技术研究颇多。工作之余，还喜欢探讨心理学，热衷出没于豆瓣和知乎。除此书外，他还译有《TCP Sockets编程》和《理解Unix进程》。

TURING 图灵程序设计丛书

Linux Shell 脚本攻略

（第2版）

[印] Shantanu Tushar
Sarath Lakshman 著
门佳 译

人民邮电出版社
北京

图书在版编目（C I P）数据

Linux Shell脚本攻略 : 第2版 / （印）图沙尔（Tushar, S.），（印）拉克什曼（Lakshman, S.）著 ; 门佳译. -- 北京 : 人民邮电出版社, 2014.1
（图灵程序设计丛书）
书名原文：Linux Shell scripting cookbook, Second Edition
ISBN 978-7-115-33921-8

Ⅰ. ①L… Ⅱ. ①图… ②拉… ③门… Ⅲ. ①Linux操作系统—程序设计 Ⅳ. ①TP316.89

中国版本图书馆CIP数据核字(2013)第285766号

内 容 提 要

本书结合丰富的实际案例介绍了如何利用 shell 命令快速开发常规任务，如何凭借短短几个命令行从 Web 挖掘数据的 shell 脚本，如何通过 shell 脚本设置以太网和无线 LAN，以及如何利用少量命令的组合完成诸如文本处理、文件管理、备份等复杂的数据管理工作等。

本书面向初、中、高级 Linux 系统管理员和程序员，是编写 shell 脚本的绝佳参考资料。

◆ 著　　[印] Shantanu Tushar　Sarath Lakshman
译　　门　佳
责任编辑　丁晓昀
责任印制　焦志炜

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路11号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京　　　　印刷

◆ 开本：800×1000　1/16
印张：19.5
字数：460千字　　　　2014年1月第1版
印数：1－5 000册　　　　2014年1月北京第1次印刷

著作权合同登记号　图字：01-2013-5214号

定价：59.00元

读者服务热线：(010)51095186转600　印装质量热线：(010)81055316

反盗版热线：(010)81055315

广告经营许可证：京崇工商广字第 0021 号

版 权 声 明

献给我至爱的双亲！感谢你们教会我思考和推理的方法，让我乐观地面对生活。

——Shantanu Tushar

前　　言

GNU/Linux是世界上最强大、最灵活的操作系统之一。在现代计算领域中，它可谓无处不在，从服务器、便携式计算机、移动电话、平板电脑到超级计算机，概莫能外。尽管配备了优美、时髦的图形用户界面，但shell依然是同Linux进行交互的最灵活的方式。

除了执行单独的命令，shell还可以执行命令脚本，所以非常易于实现各种任务的自动化操作，如生成报表、发送电子邮件、进行系统维护等。本书包含各式各样的攻略，演示了命令及shell脚本的实战用法。你可以将其作为参考，或是自己编写脚本时的灵感源泉。书中涉及的任务包括文本处理、网络运维、系统管理，不一而足。

和学习其他手艺一样，熟能生巧。待你成为shell脚本行家之日，方能完全发挥并驾驭shell的真力。本书会告诉你如何达成这一目标！

本书内容

第1章：小试牛刀。作为用于理解Bash的基本概念及特性的入门章节，这一章讨论了终端打印、数学运算以及其他一些简单的Bash功能。

第2章：命令之乐。这一章展示了GNU/Linux中的常用命令，历数了用户可能会遇到或是可以善加利用的各种实践用例。除了基础命令之外，在这一版中还讨论了加密散列命令（cryptographic hashing command）和尽可能并行执行命令的方法。

第3章：以文件之名。这一章包含了与文件及文件系统相关的多条攻略，讲解了如何生成大体积文件、将文件系统写入文件、挂载文件、创建ISO镜像。我们还探讨了查找并删除重复文件、统计文件行数、收集文件详细信息等操作。

第4章：让文本飞。这一章结合任务实例讲解了GNU/Linux下大部分命令行文本处理工具。此外还包含了一些补充内容，详细介绍了正则表达式以及`sed`/`awk`命令。本章对多数常见的文本处理任务逐一给出了解决方案。这些都是实战中不可不知的技巧。

第5章：一团乱麻？没这回事。这一章包含了多个同互联网服务相关的shell脚本，旨在帮助读者理解如何使用shell脚本与Web进行交互，以实现诸如Web页面数据采集、解析等任务的自动

化操作。讲解了以POST和GET方式发送网页，提交客户数据到服务器的方法。在这一版中，就Twitter这类服务，采用了一种全新的授权机制——OAuth。

第6章：B计划。这一章演示了用于数据备份、归档、压缩等若干命令。除了更快的压缩技术，这一版还讨论了如何创建全盘镜像。

第7章：无网不利。这一章涵盖了Linux环境下的联网实践以及若干有助于编写网络shell脚本的命令。首先介绍了一些网络基础知识，随后讲解了`ssh`的用法，这可算得上是现代GNU/Linux系统中最强大的命令之一。除此之外，我们还讨论了高级端口转发、设置原始通信信道（raw communication channel）、防火墙配置等内容。

第8章：当个好管家。这一章介绍了Linux系统活动监视相关的攻略以及日志记录和报表生成，讲解了计算磁盘使用情况、监视用户访问、CPU占用等任务。在这一版中，我们还会讲述优化电源使用、检查磁盘和文件系统错误的方法。

第9章：管理重任。这一章涉及一系列系统管理方面的实战攻略，讲解了各种用于收集系统详细信息的命令以及使用脚本进行用户管理的方法，还讨论了大图片缩放、通过shell访问MySQL数据库。在这一版中，我们还会学习在不借助窗口管理器的情况下，如何使用GNU Screen来管理多个终端。

阅读本书要求

只要具备任何一种GNU/Linux平台的一般性使用经验，都能很轻松地阅读本书。我们已竭尽所能地确保书中的所有例子清晰明了、简单易懂。在Linux平台下学习的好奇心是你阅读本书所需的唯一条件。我们为你提供了循序渐进的辅导，助你解决书中有关脚本编写的难题。为了运行并测试书中的例子，我们推荐安装Ubuntu/Debian Linux。当然，其他的Linux发行版也足以胜任绝大多数任务。你会发现就编写shell脚本来说，本书绝对是一份通俗易懂的参考资料，同时也是一位助你编写高效脚本的良师益友。

本书读者对象

如果你是一位初中级用户，希望通过掌握快速编写脚本的技巧来完成各类事务处理，而又不愿去逐页翻阅手册，那么本书就是写给你的。你不用了解任何shell脚本或Linux的工作原理，只需要参照书中类似的例子和描述就可以动手了。对于中高级用户以及系统管理员或程序员而言，本书则是在编码过程中寻求问题解决之道的一份绝佳参考资料。

本书约定

本书用多种不同格式的文本来区分不同种类的信息。下面是各类格式的例子及其所代表的含义。

正文中的代码片段像这样显示："我们创建了名为`repeat`的函数，它包含了一个无限`while`循环，该循环尝试运行被作为函数参数（通过`$@`访问）传入的命令。"

代码块以如下形式显示：

```
if  [ $var -eq 0 ]; then echo "True"; fi
```

还可以写成：

```
if  test $var -eq 0 ; then echo "True"; fi
```

如果我们希望你注意代码块的某部分，那么会使用粗体显示相关的代码行或条目：

```
while  read line;
do  something
done  < filename
```

命令行输入或输出写成如下形式：

```
# mkdir /mnt/loopback
# mount -o loop loopbackfile.img /mnt/loopback
```

新术语和**重要的词句**显示为粗体。

警告或重要的提示出现在这里。

建议和窍门则会以这种方式出现。

读者反馈

十分欢迎读者提供反馈意见。我们想知道你对本书的看法：喜欢哪些部分，不喜欢哪些部分。这些反馈对于协助我们编写出真正对读者有所裨益的书至关重要。①

你只需要向feedback@packtpub.com发送电子邮件，并在邮件标题中注明书名即可。如果你在某

① 读者也可以登录图灵社区，在本书的页面（ituring.com.cn/book/1170）下提交反馈意见、勘误以及下载本书示例代码。——编者注

方面有所专长并且愿意参与图书编写或出版，请参阅我们的作者指南www.packtpub. com/authors。

客户支持

现在你已经拥有了这本由Packt出版的图书，为了让此书尽可能地物有所值，我们还为你提供了其他诸多方面的服务。

下载示例代码

你可以在http://www.packtput.com下载本书及你所购买的所有Packt图书的示例代码。如果你是在其他地方购买的本书的英文版，可以访问http://www.packtput.com/support并注册，示例代码将用电子邮件发送给你。

勘误

尽管我们已经竭尽全力确保本书内容准确，但错误终难避免。如果你发现了书中的任何错误，无论是出现在正文还是代码中的，我们都非常乐于见到你将错误提交给我们。这样不仅能够减少其他读者的困惑，还能帮助我们改进本书后续版本的质量。如果需要提交勘误，请访问http://www.packtpub.com/submit-errata，选择相应的书名，单击**勘误提交表单**链接，就可以开始输入详细的勘误信息了。一旦勘误得到确认，我们将接受你的提交，同时勘误内容也将被上传到我们的网站，或者被添加到对应书目勘误区的现有勘误表中。所有图书当前的勘误都可以通过http://www.packtpub.com/support来查看。

举报盗版

各种媒体在网上一直饱受版权侵害的困扰。Packt坚持严格保护版权和授权。如果你在网上发现我社图书的任何形式的盗版，请立即为我们提供地址或网站名称，以便我们采取进一步的措施。

请将疑似侵权的网站链接发送至copyright@packtpub.com。

非常感谢你对保护作者知识产权所做的工作，我们将竭诚为读者提供有价值的内容。

疑难解答

如果你对本书的某方面抱有疑问，请通过questions@packtpub.com联系我们，我们会尽力为你解决。

目　录

第 1 章 小试牛刀

本章内容

- 终端打印
- 玩转变量与环境变量
- 使用函数填加环境变量
- 通过shell进行数学运算
- 玩转文件描述符与重定向
- 数组和关联数组
- 使用别名
- 获取终端信息
- 获取、设置日期及延时
- 调试脚本
- 函数和参数
- 将命令序列的输出读入变量
- 以不按回车键的方式获取字符"n"
- 运行命令直至执行成功
- 字段分隔符和迭代器
- 比较与测试

1.1 简介

诸多类Unix操作系统的设计令人惊叹。即便是在数十年后的今天，Unix式的操作系统架构仍是有史以来的最佳设计之一。这种架构最重要的一个特性就是命令行界面或shell。shell环境使得用户能与操作系统的核心功能进行交互。术语**脚本**更多涉及的便是这种环境。编写脚本通常使用某种基于解释器的编程语言。而shell脚本不过就是一些文件，我们能将一系列需要执行的命令写入其中，然后通过shell来执行。

本书使用的是Bash（Bourne Again Shell），它是目前大多数GNU/Linux系统默认的shell环境。鉴于GNU/Linux作为基于Unix式架构最杰出操作系统的地位，书中大部分案例和讨论都假定是在Linux系统环境下进行的。

本章的主要目的是让读者了解shell环境并熟悉shell的基本特性。命令都是在shell终端中输入并执行。打开终端后，就会出现一个提示符。其形式通常如下：

```
username@hostname$
```

或者

```
root@hostname #
```

要么就简单地以$或#表示。

$表示普通用户，#表示管理员用户root。root是Linux系统中权限最高的用户。

> 以root用户（管理员）的身份直接使用shell来执行任务可不是个好主意。因为如果shell具备较高的权限，命令中出现的输入错误有可能造成更严重的破坏。所以推荐使用普通用户登录系统（使用$来表明这种身份，root登录时要使用#），然后借助sudo这类工具来运行特权命令。使用sudo <command> <arguments>这种形式执行命令的效果和root一样。

shell脚本通常是一个以shebang①起始的文本文件，如下所示：

```
#!/bin/bash
```

shebang是一个文本行，其中#!位于解释器路径之前。/bin/bash是Bash的解释器命令路径。

有两种运行脚本的方式。一种是将脚本作为bash的命令行参数，另一种是授予脚本执行权限，将其变为可执行文件。

将脚本作为命令行参数时的运行方式如下（#号后面的文本是注释，不必输入到命令行中）：

```
$ bash script.sh #假设脚本位于当前目录下
```

或者

```
$ bash /home/path/script.sh #使用script.sh的完整路径
```

如果将脚本作为bash的命令行参数来运行，那么就用不着脚本中的shebang了。

要是有需要的话，可以利用shebang来实现脚本的独立运行。对此必须设置脚本的可执行权限，这样它就可以使用位于#!之后的解释器路径来运行了。就像这样：

```
$ chmod a+x script.sh
```

该命令赋予所有用户script.sh文件的可执行权限。这个脚本能以下列方式执行：

```
$ ./script.sh #./表示当前目录
```

或者

① shebang这个词其实是两个字符名称的组合。在Unix的行话里，用sharp或hash（有时候是mesh）来称呼字符“#”，用bang来称呼惊叹号“!”，因而shebang合起来就代表了这两个字符。详情请参考：http://en.wikipedia.org/wiki/Shebang_(Unix)。（注：书中脚注均为译者注。）

```
$ /home/path/script.sh #使用脚本的完整路径
```

内核会读取脚本的首行并注意到shebang为`#!/bin/bash`。它识别出/bin/bash并在内部像这样执行该脚本：

```
$ /bin/bash script.sh
```

当启动shell时，它一开始会执行一组命令来定义诸如提示文本、颜色等各类设置。这组命令来自位于用户主目录中的脚本文件~/.bashrc（对于登录shell则是~/.bash_profile）。Bash还维护了一个历史记录文件~/.bash_history，用于保存用户运行过的命令。

> ~表示主目录，它通常是`/home/user`，其中`user`是用户名，如果是root用户，则为`/root`。
>
> 登录shell是登录主机后获得的那个shell。如果登录图形界面环境（比如GNOME、KDE等）后打开了一个shell，就不是登录shell。

在Bash中，每个命令或是命令序列是通过使用分号或换行符来分隔的。比如：

```
$ cmd1 ; cmd2
```

它等同于：

```
$ cmd1
$ cmd2
```

字符#指明注释的开始。

注释部分以#为起始，一直延续到行尾。注释行通常用于为代码提供注释信息，或者停止执行某行代码。[1]

现在让我们继续讨论基本特性。

1.2 终端打印

终端是交互式工具，用户可以通过它与shell环境进行交互。在终端中打印文本是绝大多数shell脚本和工具日常需要执行的基本任务。在这则攻略中我们会看到，可以使用各种方法，采用各种格式进行打印。

① shell不执行脚本中的任何注释部分。

1.2.1　实战演练

echo是用于终端打印的基本命令。

在默认情况下，echo在每次调用后会添加一个换行符。

```
$ echo "Welcome to Bash"
Welcome to Bash
```

只需要使用带双引号的文本，结合echo命令就可以将该文本在终端中打印出来。类似地，不带双引号的文本也可以得到同样的输出结果：

```
$ echo Welcome to Bash
Welcome to Bash
```

使用单引号也可以完成同样的任务：

```
$ echo 'text in quotes'
```

这些方法看起来相似，但各有一些特殊用途和副作用。思考下面这行命令：

```
$ echo "cannot include exclamation - ! within double quotes"
```

这条命令将会返回：

```
bash: !: event not found error
```

因此，如果需要打印!，那就不要将其放入双引号中，或者你可以在其之前加上一个特殊的转义字符（\）将!转义，就像这样：

```
$ echo Hello world !
```

或者

```
$ echo 'Hello world !'
```

或者

```
$ echo "Hello world \!" #将转义字符放在前面
```

每种方法的副作用如下所述。

- 使用不带引号的echo时，没法在所要显示的文本中使用分号（;），因为分号在Bash shell中被用作命令定界符。
- 以echo hello;hello为例，echo hello被视为一个命令，第二个hello则被视为另一个命令。
- 变量替换在单引号中无效，在下一则攻略中会详细讨论。

另一个可用于终端打印的命令是`printf`。`printf`使用的参数和C语言中的`printf`函数一样。例如：

```
$ printf "Hello world"
```

`printf`使用引用文本或由空格分隔的参数。我们可以在`printf`中使用格式化字符串，还可以指定字符串的宽度、左右对齐方式等。在默认情况下，`printf`并不像echo命令一样会自动添加换行符，我们必须在需要的时候手动添加，比如在下面的脚本中：

```
#!/bin/bash
#文件名：printf.sh

printf  "%-5s %-10s %-4s\n" No Name  Mark
printf  "%-5s %-10s %-4.2f\n" 1 Sarath 80.3456
printf  "%-5s %-10s %-4.2f\n" 2 James 90.9989
printf  "%-5s %-10s %-4.2f\n" 3 Jeff 77.564
```

我们会得到如下格式化的输出：

```
No     Name        Mark
1      Sarath      80.35
2      James       91.00
3      Jeff        77.56
```

1.2.2　工作原理

`%s`、`%c`、`%d`和`%f`都是格式替换符（format substitution character），其所对应的参数可以置于带引号的格式字符串之后。

`%-5s`指明了一个格式为左对齐且宽度为5的字符串替换（`-`表示左对齐）。如果不用`-`指定对齐方式，字符串就采用右对齐形式。宽度指定了保留给某个变量的字符数。对`Name`而言，保留宽度是10。因此，任何`Name`字段的内容都会被显示在10字符宽的保留区域内，如果内容不足10个字符，余下的则以空格符填充。

对于浮点数，可以使用其他参数对小数部分进行舍入。

对于`Mark`字段，将其格式化为`%-4.2f`，其中.2指定保留2个小数位。注意，在每行格式字符串后都有一个换行符（`\n`）。

1.2.3　补充内容

使用echo和`printf`的命令选项时，要确保选项应该出现在命令行内所有字符串之前，否则Bash会将其视为另外一个字符串。

1. 在echo中转义换行符

在默认情况下，echo会将一个换行符追加到输出文本的尾部。可以使用选项-n来忽略结尾的换行符。echo同样接受双引号字符串内的转义序列作为参数。如果需要使用转义序列，则采用echo -e "包含转义序列的字符串"这种形式。例如：

```
echo -e "1\t2\t3"
1   2   3
```

2. 打印彩色输出

在终端中生成彩色输出相当好玩，我们可以使用转义序列来实现。

每种颜色都有对应的颜色码。比如：重置=0，黑色=30，红色=31，绿色=32，黄色=33，蓝色=34，洋红=35，青色=36，白色=37。

要打印彩色文本，可输入如下命令：

```
echo -e "\e[1;31m This is red text \e[0m"
```

\e[1;31将颜色设为红色，\e[0m将颜色重新置回。只需要将31替换成想要的颜色码就可以了。

要设置彩色背景，经常使用的颜色码是：重置=0，黑色=40，红色=41，绿色=42，黄色=43，蓝色=44，洋红=45，青色=46，白色=47。

要打印彩色文本，可输入如下命令：

```
echo -e "\e[1;42m Green Background \e[0m"
```

1.3　玩转变量和环境变量

变量是任何一种编程语言都必不可少的组成部分，用于存放各类数据。脚本语言通常不需要在使用变量之前声明其类型。只需要直接赋值就可以了。在Bash中，每一个变量的值都是字符串。无论你给变量赋值时有没有使用引号，值都会以字符串的形式存储。有一些特殊的变量会被shell环境和操作系统环境用来存储一些特别的值，这类变量就被称为**环境变量**。让我们来看一些实例。

1.3.1　预备知识

变量采用常见的命名方式进行命名。当应用程序执行时，它接收一组环境变量。可以使用env命令查看所有与终端相关的环境变量。对于进程来说，其运行时的环境变量可以使用下面的命令来查看：

```
cat /proc/$PID/environ
```

其中，将PID设置成相关进程的进程ID（PID总是一个整数）。

假设有一个叫做gedit的应用程序正在运行。我们可以使用pgrep命令获得gedit的进程ID：

```
$ pgrep gedit
12501
```

那么，你就可以通过以下命令获得与该进程相关的环境变量：

```
$ cat /proc/12501/environ
GDM_KEYBOARD_LAYOUT=usGNOME_KEYRING_PID=1560USER=slynuxHOME=/home/slynux
```

> 环境变量远不止这些，只是由于对页面篇幅的限制，在这里删除了其他很多环境变量。

上面的命令返回一个包含环境变量以及对应变量值的列表。每一个变量以name=value的形式来描述，彼此之间由null字符（\0）分隔。如果你将\0替换成\n，那么就可以将输出重新格式化，使得每一行显示一组“变量=值”。替换可以使用tr命令来实现：

```
$ cat /proc/12501/environ  | tr '\0' '\n'
```

现在，让我们看看怎样对变量和环境变量进行赋值及处理。

1.3.2 实战演练

变量可以通过以下方式进行赋值：

```
var=value
```

var是变量名，value是赋给变量的值。如果value不包含任何空白字符（例如空格），那么它就不需要使用引号进行引用，否则必须使用单引号或双引号。

注意，var = value不同于var=value。把var=value写成var = value是一个常见的错误，但前者是赋值操作，后者则是相等操作。

在变量名之前加上$前缀就可以打印出变量的内容：

```
var="value" #给变量var赋值

echo $var
```

或者

```
echo ${var}
```

输出如下：

```
value
```

我们可以在printf或echo命令的双引号中引用变量值。

```
#!/bin/bash
#文件名:variables.sh
fruit=apple
count=5
echo "We have $count ${fruit}(s)"
```

输出如下：

```
We have 5 apple(s)
```

环境变量是未在当前进程中定义，而从父进程中继承而来的变量。例如环境变量HTTP_PROXY，它定义了互联网连接应该使用哪个代理服务器。

该环境变量通常被设置成：

```
HTTP_PROXY=192.168.1.23:3128
export HTTP_PROXY
```

export命令用来设置环境变量。至此之后，从当前shell脚本执行的任何应用程序都会继承这个变量。我们可以按照自己的需要，在执行的应用程序或者shell脚本中导出特定的变量。在默认情况下，有很多标准环境变量可供shell使用。

PATH就是其中之一。通常，变量PATH包含：

```
$ echo $PATH
/home/slynux/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr
/games
```

在给出所要执行的命令后，shell会自动在PATH环境变量所包含的目录列表中（各目录路径之间以冒号分隔）查找对应的可执行文件。PATH通常定义在/etc/environment或/etc/profile或~/.bashrc中。如果需要在PATH中添加一条新路径，可以使用：

```
export PATH="$PATH:/home/user/bin"
```

也可以使用

```
$ PATH="$PATH:/home/user/bin"
$ export PATH

$ echo $PATH
/home/slynux/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr
/games:/home/user/bin
```

这样，我们就将/home/user/bin添加到了PATH中。

还有一些众所周知的环境变量：HOME、PWD、USER、UID、SHELL等。

> 使用单引号时，变量不会被扩展（expand），将依照原样显示。这意味着：
>
> `$ echo '$var' will print $var`
>
> 但如果变量var已经定义过，那么`$ echo "$var"`会打印出该变量的值；如果没有定义过，则什么都不打印。

1.3.3 补充内容

让我们再多看些有关标准变量和环境变量的技巧。

1. 获得字符串长度

可以用下面的方法获得变量值的长度：

```
length=${#var}
```

例如：

```
$ var=12345678901234567890$
echo ${#var}
20
```

`length`就是字符串所包含的字符数。

2. 识别当前所使用的shell

可以用下面的方法获知当前使用的是哪种shell：

```
echo $SHELL
```

也可以用

```
echo $0
```

例如：

```
$ echo $SHELL
/bin/bash

$ echo $0
/bin/bash
```

3. 检查是否为超级用户

`UID`是一个重要的环境变量，可以用于检查当前脚本是以超级用户还是以普通用户的身份运行的。例如：

```
If [ $UID -ne 0 ]; then
    echo Non root user. Please run as root.
else
    echo Root user
fi
```

root用户的UID是0。

4. 修改Bash提示字符串（**username@hostname:~$**）

当我们打开终端或是运行shell时，会看到类似于user@hostname:/home/$的提示字符串。不同GNU/Linux发布版中的提示及颜色略有不同。我们可以利用PS1环境变量来定制提示文本。默认的shell提示文本是在文件~/.bashrc中的某一行设置的。

- 可以使用如下命令列出设置变量PS1的那一行：

```
$ cat ~/.bashrc | grep PS1
PS1='${debian_chroot:+($debian_chroot)}\u@\h:\w\$ '
```

- 如果要设置一个定制的提示字符串，可以输入：

```
slynux@localhost: ~$ PS1="PROMPT>"
PROMPT> Type commands here #提示字符串已经改变
```

- 我们可以利用类似\e[1;31的特定转义序列来设置彩色的提示字符串（参考1.2节的内容）。

还有一些特殊的字符可以扩展成系统参数。例如：\u可以扩展为用户名，\h可以扩展为主机名，而\w可以扩展为当前工作目录。

1.4　使用函数添加环境变量

环境变量通常用于存储路径列表，这些路径用于搜索可执行文件、库文件等。例如$PATH、$LD_LIBRARY_PATH，它们通常看起来像这样：

```
PATH=/usr/bin;/bin
LD_LIBRARY_PATH=/usr/lib;/lib
```

这意味着只要shell需要运行二进制可执行文件时，它会首先查找/usr/bin，然后是/bin。

当你必须使用源代码编译生成程序并将其安装到某个特定路径中时，有项极其常见的任务就是将该程序的bin目录加入PATH环境变量。假设我们要将myapp安装到/opt/myapp，它的二进制文件在bin目录中，库文件在lib目录中。

1.4.1 实战演练

实现方法如下：

```
export PATH=/opt/myapp/bin:$PATH
export LD_LIBRARY_PATH=/opt/myapp/lib;$LD_LIBRARY_PATH
```

PATH和LD_LIBRARY_PATH现在看起来应该像这样：

```
PATH=/opt/myapp/bin:/usr/bin:/bin
LD_LIBRARY_PATH=/opt/myapp/lib:/usr/lib;/lib
```

不过我们可以把下面的函数加入.bashrc-，让一切变得更轻松些：

```
prepend() { [ -d "$2" ] && eval $1=\"$2':'\$$1\" && export $1; }
```

像下面这样来使用该函数：

```
prepend PATH /opt/myapp/bin
prepend LD_LIBRARY_PATH /opt/myapp/lib
```

1.4.2 工作原理

我们定义了名为prepend()的函数，它首先检查该函数第二个参数所指定的目录是否存在。如果存在，eval表达式将第一个参数所指定的变量值设置成第二个参数的值加上“:”（路径分隔符），随后再跟上首个参数的原始值。

不过，有一点需要留意。在进行添加时，如果变量为空，则会在末尾留下一个“:”。要解决这个问题，可以将该函数再进行一些修改：

```
prepend() { [ -d "$2" ] && eval $1=\"$2\$\{$1:+':'\$$1\}\" && export $1 ; }
```

> 在这个函数中，我们引入了一种shell参数扩展的形式：
>
> `${parameter:+expression}`
>
> 如果parameter有值且不为空，则使用expression的值。

通过这次修改，在追加环境变量时，当且仅当旧值存在，才会增加。

1.5 使用 shell 进行数学运算

无论哪种编程语言都少不了算术操作，在这则攻略中，我们将会研究在shell中进行算术运算的各种方法。

1.5.1 预备知识

在Bash shell环境中，可以利用let、(( ))和[]执行基本的算术操作。而在进行高级操作时，expr和bc这两个工具也会非常有用。

1.5.2 实战演练

(1) 可以用普通的变量赋值方法定义数值，这时，它会被存储为字符串。然而，我们可以用一些方法使它能像数字一样进行运算。

```
#!/bin/bash
no1=4;
no2=5;
```

(2) let命令可以直接执行基本的算术操作。当使用let时，变量名之前不需要再添加$，例如：

```
let result=no1+no2
echo $result
```

- 自加操作

```
$ let no1++
```

- 自减操作

```
$ let no1--
```

- 简写形式

```
let no+=6
let no-=6
```

它们分别等同于let no=no+6和let no=no-6。

- 其他方法

操作符[]的使用方法和let命令类似：

```
result=$[ no1 + no2 ]
```

在[]中也可以使用$前缀，例如：

```
result=$[ $no1 + 5 ]
```

也可以使用(())，但使用(())时，变量名之前需要加上$：

```
result=$(( no1 + 50 ))
```

expr同样可以用于基本算术操作：

```
result=`expr 3 + 4`
result=$(expr $no1 + 5)
```

以上这些方法只能用于整数运算，而不支持浮点数。

(3) bc是一个用于数学运算的高级工具，这个精密计算器包含了大量的选项。我们可以借助它执行浮点数运算并应用一些高级函数：

```
echo "4 * 0.56" | bc
2.24

no=54;
result=`echo "$no * 1.5" | bc`
echo $result
81.0
```

其他参数可以置于要执行的具体操作之前，同时以分号作为定界符，通过stdin传递给bc。

- **设定小数精度**。在下面的例子中，参数scale=2将小数位个数设置为2。因此，bc将会输出包含两个小数位的数值。

```
echo "scale=2;3/8" | bc
0.37
```

- **进制转换**。用bc可以将一种进制系统转换为另一种。来看看如何将十进制转换成二进制，然后再将二进制转换回十进制：

```
#!/bin/bash
用途：数字转换

no=100
echo "obase=2;$no" | bc
1100100
no=1100100
echo "obase=10;ibase=2;$no" | bc
100
```

- **计算平方以及平方根**。

```
echo "sqrt(100)" | bc #Square root
echo "10^10" | bc #Square
```

1.6 玩转文件描述符及重定向

文件描述符是与文件输入、输出相关联的整数。它们用来跟踪已打开的文件。最常见的文件描述符是stdin、stdout和stderr。我们甚至可以将某个文件描述符的内容重定向到另一个文件描述符中。下面给出一些对文件描述符进行操作和重定向的例子。

1.6.1 预备知识

在编写脚本的时候会频繁使用标准输入（stdin）、标准输出（stdout）和标准错误（stderr）。通过内容过滤将输出重定向到文件是我们平日里的基本任务之一。当命令输出文本时，这些输出文本有可能是错误信息，也可能是正常的（非错误的）输出信息。单靠查看输出的文本本身，我们没法区分哪些是正常，哪些是错误。不过可以通过文件描述符来解决这个问题，将那些与特定描述符关联的文本提取出来。

文件描述符是与某个打开的文件或数据流相关联的整数。文件描述符0、1以及2是系统预留的。

- **0** —— stdin（标准输入）。
- **1** —— stdout（标准输出）。
- **2** —— stderr（标准错误）。

1.6.2 实战演练

(1) 用下面的方法可以将输出文本重定向或保存到一个文件中：

```
$ echo "This is a sample text 1" > temp.txt
```

这种方法通过截断文件的方式，将输出文本存储到文件temp.txt中，也就是说在把echo命令的输出写入文件之前，temp.txt中的内容首先会被清空。

(2) 将文本追加到目标文件中，看下面的例子：

```
$ echo "This is sample text 2" >> temp.txt
```

(3) 查看文件内容：

```
$ cat temp.txt
This is sample text 1
This is sample text 2
```

(4) 来看看什么是标准错误以及如何对它重定向。当命令输出错误信息时，stderr信息就会被打印出来。考虑下面的例子：

```
$ ls +
ls: cannot access +: No such file or directory
```

这里，+是一个非法参数，因此将返回错误信息。

> **成功和不成功的命令**
>
> 当一个命令发生错误并退回时，它会返回一个非0的退出状态；而当命令成功完成后，它会返回数字0。退出状态可以从特殊变量$?中获得（在命令执行之后立刻运行echo $?，就可以打印出退出状态）。

下面的命令会将stderr文本打印到屏幕上，而不是文件中（而且因为并没有stdout的输出，所以out.txt没有内容）：

```
$ ls + > out.txt
ls: cannot access +: No such file or directory
```

下面的命令中，我们将stderr重定向到out.txt：

```
$ ls + 2> out.txt #正常运行
```

你可以将stderr单独重定向到一个文件，将stdout重定向到另一个文件：

```
$ cmd 2>stderr.txt 1>stdout.txt
```

还可以利用下面这个更好的方法将stderr转换成stdout，使得stderr和stdout都被重定向到同一个文件中：

```
$ cmd 2>&1 output.txt
```

或者这样：

```
$ cmd &> output.txt
```

(5) 有时候，在输出中可能包含一些不必要的信息（比如调试信息）。如果你不想让终端中充斥着有关stderr的繁枝末节，那么你可以将stderr的输出重定向到 /dev/null，保证一切都会被清除得干干净净。假设我们有3个文件，分别是a1、a2、a3。但是普通用户对文件a1没有“读–写–执行”权限。如果需要打印文件名以a起始的所有文件的内容，可以使用cat命令。设置一些测试文件：

```
$ echo a1 > a1
$ cp a1 a2 ; cp a2 a3;
$ chmod 000 a1  #清除所有权限
```

尽管可以使用通配符（a*）显示所有的文件内容，但是系统会显示一个出错信息，因为对文件a1没有可读权限。

```
$ cat a*
cat: a1: Permission denied
a1
a1
```

其中，cat: a1: Permission denied属于stderr。我们可以将stderr信息重定向到一个文件中，而stdout仍然保持不变。考虑如下代码：

```
$ cat a* 2> err.txt # stderr被重定向到err.txt
a1
a1

$ cat err.txt
cat: a1: Permission denied
```

观察下面的代码：

```
$ cmd 2>/dev/null
```

当对如果对stderr或stdout进行重定向，被重定向的文本会传入文件。因为文本已经被重定向到文件中，也就没剩下什么东西可以通过管道（|）传给接下来的命令，而这些命令是通过stdin进行接收的。

(6) 但是有一个方法既可以将数据重定向到文件，还可以提供一份重定向数据的副本作为后续命令的stdin。这一切都可以使用tee来实现。举个例子：要在终端中打印stdout，同时将它重定向到一个文件中，那么可以这样使用tee:

```
command | tee FILE1 FILE2
```

在下面的代码中，tee命令接收到来自stdin的数据。它将stdout的一份副本写入文件out.txt，同时将另一份副本作为后续命令的stdin。命令cat -n将从stdin中接收到的每一行数据前加上行号并写入stdout:

```
$ cat a* | tee out.txt | cat -n
cat: a1: Permission denied
     1a1
     2a1
```

查看out.txt的内容：

```
$ cat out.txt
a1
a1
```

注意，cat: a1: Permission denied 并没有在文件内容中出现。这是因为这些信息属于stderr，而tee只能从stdin中读取。

默认情况下，tee命令会将文件覆盖，但它提供了一个-a选项，用于追加内容。例如：$ cat a* | tee -a out.txt | cat -n。

带有参数的命令可以写成：command FILE1 FILE2…或者就简单的使用command FILE。

(7) 我们可以使用stdin作为命令参数。只需要将-作为命令的文件名参数即可：

```
$ cmd1 | cmd2 | cmd -
```

例如：

```
$ echo who is this | tee -
who is this
who is this
```

也可以将 /dev/stdin作为输出文件名来代替stdin。

类似地，使用 /dev/stderr代表标准错误，/dev/stdout代表标准输出。这些特殊的设备文件分别对应stdin、stderr和stdout。

1.6.3　工作原理

就输出重定向而言，>和>>并不相同。尽管两者可以将文本重定向到文件，但是前者会先清空文件，然后再写入内容，而后者会将内容追加到现有文件的尾部。

当使用重定向操作符时，输出内容不会在终端打印，而是被导向文件。重定向操作符默认使用标准输出。如果想使用特定的文件描述符，你必须将描述符编号置于操作符之前。

>等同于1>；对于>>来说，情况也类似（即>>等同于1>>）。

处理错误时，来自stderr的输出被丢弃到文件/dev/null中。/dev/null是一个特殊的设备文件，它接收到的任何数据都会被丢弃。null设备通常也被称为**黑洞**，因为凡是到这儿的数据都将一去不返。

1.6.4　补充内容

从stdin读取输入的命令能以多种方式接收数据。也可以用cat和管道来指定我们自己的文件描述符，例如：

```
$ cat file | cmd
$ cmd1 | cmd
```

1. 将文件重定向到命令

借助重定向，我们可以像使用stdin那样从文件中读取数据：

```
$ cmd < file
```

2. 将脚本内部的文本块进行重定向

有时候，我们需要对文本块（多行文本）进行重定向，就像对标准输入做的那样。考虑一个特殊情况：源文本就位于shell脚本中。一个实用的例子是向log文件中写入头部数据，可以按照下面的方法完成：

```
#!/bin/bash
cat<<EOF>log.txt
LOG FILE HEADER
This is a test log file
Function: System statistics
EOF
```

在cat <<EOF>log.txt与下一个EOF行之间的所有文本行都会被当做stdin数据。log.txt文件的内容打印如下：

```
$ cat log.txt
LOG FILE HEADER
This is a test log file
Function: System statistics
```

3. 自定义文件描述符

文件描述符是一种用于访问文件的抽象指示器（abstract indicator）。存取文件离不开被称为“文件描述符”的特殊数字。0、1和2分别是stdin、stdout和stderr的预留描述符编号。

我们可以使用exec命令创建自己的文件描述符。如果你对用其他编程语言进行文件编程非常熟悉，你可能已经注意到了文件打开模式。通常会用到3种模式。

- 只读模式。
- 截断写入模式。
- 追加写入模式。

< 操作符用于从文件中读取至stdin。> 操作符用于截断模式的文件写入（数据在目标文件内容被截断之后写入）。>>操作符用于追加模式的文件写入。（数据被添加到文件的现有内容中，而且该目标文件中原有的内容不会丢失。）文件描述符可以用以上三种模式中的任意一种来创建。

创建一个文件描述符进行文件读取：

```
$ exec 3<input.txt #使用文件描述符3打开并读取文件
```

我们可以这样使用它：

```
$ echo this is a test line > input.txt
$ exec 3<input.txt
```

现在你就可以在命令中使用文件描述符3了。例如：

```
$ cat<&3
this is a test line
```

如果要再次读取，我们就不能继续使用文件描述符3了，而是需要用exec重新分配文件描述符3来进行二次读取。

创建一个文件描述符用于写入（截断模式）：

```
$ exec 4>output.txt #打开文件进行写入
```

例如：

```
$ exec 4>output.txt
$ echo newline >&4
$ cat output.txt
newline
```

创建一个文件描述符用于写入（追加模式）：

```
$ exec 5>>input.txt
```

例如：

```
$ exec 5>>input.txt
$ echo appended line >&5
$ cat input.txt
newline
appended line
```

1.7 数组和关联数组

数组是shell脚本非常重要的组成部分，它借助索引将多个独立的数据存储为一个集合。普通数组只能使用整数作为数组索引。Bash也支持关联数组，它可以使用字符串作为数组索引。在很多情况下，采用字符串式索引更容易理解，这时候关联数组就派上用场了。在这里，我们会看到普通数组和关联数组的用法。

1.7.1 预备知识

Bash从4.0版本之后才开始支持关联数组。

1.7.2　实战演练

(1) 定义数组的方法有很多种。可以在单行中使用一列值来定义一个数组：

```
array_var=(1 2 3 4 5 6)
#这些值将会存储在以0为起始索引的连续位置上
```

另外，还可以将数组定义成一组“索引-值”：

```
array_var[0]="test1"
array_var[1]="test2"
array_var[2]="test3"
array_var[3]="test4"
array_var[4]="test5"
array_var[5]="test6"
```

(2) 打印出特定索引的数组元素内容：

```
echo ${array_var[0]}
test1
index=5
echo ${array_var[$index]}
test6
```

(3) 以清单形式打印出数组中的所有值：

```
$ echo ${array_var[*]}
test1 test2 test3 test4 test5 test6
```

也可以这样使用：

```
$ echo ${array_var[@]}
test1 test2 test3 test4 test5 test6
```

(4) 打印数组长度（即数组中元素的个数）：

```
$ echo ${#array_var[*]}
6
```

1.7.3　补充内容

关联数组从Bash 4.0版本开始被引入。借助散列技术，关联数组成为解决很多问题的有力工具。接下来就让我们一探究竟。

1. 定义关联数组

在关联数组中，我们可以用任意的文本作为数组索引。首先，需要使用声明语句将一个变量名声明为关联数组。像下面这样：

```
$ declare -A ass_array
```

声明之后，可以用两种方法将元素添加到关联数组中。

- 利用内嵌“索引-值”列表的方法，提供一个“索引-值”列表：

```
$ ass_array=([index1]=val1 [index2]=val2)
```

- 使用独立的“索引-值”进行赋值：

```
$ ass_array[index1]=val1
$ ass_array'index2]=val2
```

举个例子，试想如何用关联数组为水果制定价格：

```
$ declare -A fruits_value
$ fruits_value=([apple]='100dollars' [orange]='150 dollars')
```

用下面的方法显示数组内容：

```
$ echo "Apple costs ${fruits_value[apple]}"
Apple costs 100 dollars
```

2. 列出数组索引

每一个数组元素都有一个索引用于查找。普通数组和关联数组具有不同的索引类型。我们可以用下面的方法获取数组的索引列表：

```
$ echo ${!array_var[*]}
```

也可以使用：

```
$ echo ${!array_var[@]
```

以先前提到的`fruits_value`数组为例，运行如下命令：

```
$ echo ${!fruits_value[*]}
orange apple
```

对于普通数组，这个方法同样可行。

1.8 使用别名

别名就是一种便捷方式，以省去用户输入一长串命令序列的麻烦。下面我们会看到如何使用`alias`命令创建别名。

1.8.1 实战演练

你可以利用别名进行多种操作，如下所示。

(1) 可以按照下面的方式创建一个别名：

```
$ alias new_command='command sequence'
```

为安装命令apt-get install创建别名：

```
$ alias install='sudo apt-get install'
```

这样一来，我们就可以用install pidgin代替sudo apt-get install pidgin了。

(2) alias命令的作用只是暂时的。一旦关闭当前终端，所有设置过的别名就失效了。为了使别名设置一直保持作用，可以将它放入~/.bashrc文件中。因为每当一个新的shell进程生成时，都会执行 ~/.bashrc中的命令。

```
$ echo 'alias cmd="command seq"' >> ~/.bashrc
```

(3) 如果需要删除别名，只用将其对应的语句（如果有的话）从 ~/.bashrc中删除，或者使用unalias命令。或者使用alias example=，这会取消名为example的别名。

(4) 我们可以创建一个别名rm，它能够删除原始文件，同时在backup目录中保留副本：

```
alias rm='cp $@ ~/backup && rm $@'
```

> 创建别名时，如果已经有同名的别名存在，那么原有的别名设置将被新的设置取代。

1.8.2　补充内容

有时别名也会造成安全问题。下面来看看应该如何识别这些隐患。

对别名进行转义

alias命令能够为任何重要的命令创建别名，不过你未必总是希望用别名来执行这个命令。我们可以将希望使用的命令进行转义，从而忽略当前定义的别名。例如：

```
$ \command
```

字符\对命令实施转义，使我们可以执行原本的命令，而不是这些命令的别名替身。在不可信环境下执行特权命令，通过在命令前加上 \ 来忽略可能存在的别名设置总是一个不错的安全实践。因为攻击者可能已经将一些别有用心的命令利用别名伪装成了特权命令，借此来盗取用户输入的重要信息。

1.9　获取终端信息

编写命令行shell脚本时，总是免不了大量处理当前终端的相关信息，比如行数、列数、光标位置、密码字段等。这则攻略将帮助你学习如何采集和处理终端设置。

1.9.1　预备知识

tput和stty是两款终端处理工具。下面来看看如何用它们完成各种不同的任务。

1.9.2　实战演练

- 获取终端的行数和列数：

```
tput cols
tput lines
```

- 打印出当前终端名：

```
tput longname
```

- 将光标移动到坐标(100,100)处：

```
tput cup 100 100
```

- 设置终端背景色：

```
tputsetb n
```

其中，n可以在0到7之间取值。

- 设置文本前景色：

```
tputsetf n
```

其中，n可以在0到7之间取值。

- 设置文本样式为粗体：

```
tput bold
```

- 设置下划线的起止：

```
tput smul
tput rmul
```

- 删除从当前光标位置到行尾的所有内容：

```
tputed
```

- 在输入密码时，不应该显示输入内容。在下面的例子中，我们将看到如何使用stty来实现这一要求：

```
#!/bin/sh
#Filename: password.sh
echo -e "Enter password: "
stty -echo
read password
stty echo
echo
echo Password read.
```

> 选项-echo禁止将输出发送到终端，而选项echo则允许发送输出。

1.10　获取、设置日期和延时

很多应用程序需要以不同的格式打印日期、设置日期和时间、根据日期和时间执行操作。延时通常用于在程序执行过程中提供一段等待时间（比如1秒）。例如需要在脚本中对某项任务每隔5秒监视一次，就需要知道如何在程序中加入延时。这则攻略会告诉你怎么处理日期以及延时。

1.10.1　预备知识

我们能够以多种格式打印日期，也可以在命令行中设置日期。在类Unix系统中，日期被存储成一个整数，其大小为自**世界标准时间**（UTC）①1970年1月1日0时0分0秒②起所流逝的秒数。这种计时方式称为**纪元时**或**Unix时间**。来看看对其进行读取和设置的方式。

1.10.2　实战演练

可以使用不同的格式来读取、设置日期。实现步骤如下所示。

(1) 读取日期：

```
$ date
Thu May 20 23:09:04 IST 2010
```

(2) 打印纪元时：

```
$ date +%s
1290047248
```

① UTC（Coordinated Universal Time），又称世界标准时间或世界协调时间。UTC是以原子时秒长为基础，在时刻上尽量接近于世界时的一种时间计量系统。

② Unix认为UTC 1970年1月1日0点是纪元时间。POSIX标准推出后，这个时间也被称为POSIX时间。

我们可以从给定格式的日期串中得出对应的纪年时。在输入时有多种日期格式可供选择。如果要从系统日志中或者其他标准应用程序生成的输出中获取日期，那就完全不用烦心日期串格式的问题。要将日期串转换成纪元时，只需要这样即可实现：

```
$ date --date "Thu Nov 18 08:07:21 IST 2010" +%s
1290047841
```

选项--date用于提供日期串作为输入。但我们可以使用任意的日期格式化选项来打印输出。将日期串作为输入能够用来获知给定的日期是星期几。

例如：

```
$ date --date "Jan 20 2001" +%A
Saturday
```

表1-1是一份日期格式字符串列表。

(3) 用格式串结合 + 作为date命令的参数，可以按照你的选择打印出对应格式的日期。例如：

```
$ date "+%d %B %Y"
20 May 2010
```

(4) 设置日期和时间：

```
# date -s "格式化的日期字符串"
```

例如：

```
# date -s "21 June 2009 11:01:22"
```

(5) 有时候，我们需要检查一组命令所花费的时间，可以使用以下代码：

```
#!/bin/bash
#文件名: time_take.sh
start=$(date +%s)
commands;
statements;

end=$(date +%s)
difference=$(( end - start))
echo Time taken to execute commands is $difference seconds.
```

> 另一种方法则是使用time<scriptpath>来得到执行脚本所花费的时间。

1.10.3　工作原理

纪元时被定义为从**世界标准时间**1970年1月1日0时0分0秒起至当前时刻的总秒数，不包括闰秒[①]。当计算两个日期或两段时间的差值时，纪元时很有用处。你可以得出两个特定时间戳的纪元时间，并计算出两者之间的差值，由此就能知道两个日期之间相隔了多少秒。

利用日期格式来获得所需要的输出，可以参考表1-1。

表　1-1

日期内容	格　式
星期	`%a`（例如：Sat） `%A`（例如：Saturday）
月	`%b`（例如：Nov） `%B`（例如：November）
日	`%d`（例如：31）
固定格式日期（mm/dd/yy）	`%D`（例如：10/18/10）
年	`%y`（例如：10） `%Y`（例如：2010）
小时	`%I`或`%H`（例如：08）
分钟	`%M`（例如：33）
秒	`%S`（例如：10）
纳秒	`%N`（例如：695208515）
Unix纪元时（以秒为单位）	`%s`（例如：1290049486）

1.10.4　补充内容

编写以循环方式运行的监视脚本时，设置时间间隔是必不可少的。让我们来看看如何生成延时。

在脚本中生成延时

为了在脚本中推迟执行一段时间，可以使用`sleep`：`$ sleep no_of_seconds`.例如，下面的脚本就使用`tput`和`sleep`从0开始计数到40：

```
#!/bin/bash
#文件名：sleep.sh
echo -n Count:
tput sc

count=0;
while true;
```

```
do
    if [ $count -lt 40 ];
    then
        let count++;
        sleep 1;
        tput rc
        tput ed
        echo -n $count;
    else exit 0;
    fi
done
```

在上面的例子中，变量count初始化为0，随后每循环一次便增加1。echo语句打印出count的值。我们用tput sc存储光标位置。在每次循环中，通过恢复之前存储的光标位置，在终端中打印出新的count值。恢复光标位置的命令是tput rc。tput ed清除从当前光标位置到行尾之间的所有内容，使得旧的count值可以被清除并写入新值。循环内的1秒钟延时是通过sleep命令来实现的。

1.11 调试脚本

调试功能是每一种编程语言都应该实现的重要特性之一，当出现一些始料未及的情况时，用它来生成脚本运行信息。调试信息可以帮你弄清楚是什么原因使得程序发生崩溃或行为异常。每位系统程序员都应该了解Bash提供的调试选项。

1.11.1 实战演练

我们可以利用Bash内建的调试工具，或者按照易于调试的方式编写脚本，方法如下所示。

(1) 使用选项-x，启用shell脚本的跟踪调试功能：

```
$ bash -x script.sh
```

运行带有-x标志的脚本可以打印出所执行的每一行命令以及当前状态。注意，你也可以使用sh -x script。

(2) 使用set -x和set +x对脚本进行部分调试。例如：

```
#!/bin/bash
#文件名: debug.sh
for i in {1..6};
do
    set -x
    echo $i
    set +x
```

```
done
echo "Script executed"
```

在上面的脚本中，只会打印出echo $i的调试信息，因为使用了-x和+x对调试区域进行了限制。

(3) 前面介绍的调试手段是Bash内建的。它们通常以固定的格式生成调试信息。但是在很多情况下，我们需要以自定义格式显示调试信息。这可以通过传递 _DEBUG环境变量来建立这类调试风格。

请看下面的代码：

```
#!/bin/bash
function DEBUG()
{
    [ "$_DEBUG" == "on" ] && $@ || :
}
for i in {1..10}
do
    DEBUG echo $i
done
```

可以将调试功能置为"on"来运行上面的脚本：

```
$ _DEBUG=on ./script.sh
```

我们在每一个需要打印调试信息的语句前加上DEBUG。如果没有把 _DEBUG=on传递给脚本，那么调试信息就不会打印出来。在Bash中，命令“:”告诉shell不要进行任何操作。

1.11.2　工作原理

-x标识将脚本中执行过的每一行都输出到stdout。不过，我们也可能只关注脚本某些部分的命令及参数的打印输出。针对这种情况，可以在脚本中使用set builtin来启用或禁止调试打印。

- set -x：在执行时显示参数和命令。
- set +x：禁止调试。
- set -v：当命令进行读取时显示输入。
- set +v：禁止打印输入。

1.11.3　补充内容

还有其他脚本调试的便捷方法，我们甚至可以巧妙地利用shebang来进行调试。

shebang的妙用

把shebang从`#!/bin/bash`改成 `#!/bin/bash -xv`，这样一来，不用任何其他选项就可以启用调试功能了。

1.12 函数和参数

和其他脚本语言一样，Bash同样支持函数。让我们看看它是如何定义和使用函数的。

1.12.1 实战演练

我们可以创建执行特定任务的函数，也可以创建能够接受参数的函数。方法如下所示。

(1) 定义函数：

```
function fname()
{
    statements;
}
```

或者：

```
fname()
{
    statements;
}
```

(2) 只需要使用函数名就可以调用某个函数：

```
$ fname ; #执行函数
```

(3) 参数可以传递给函数，并由脚本进行访问：

```
fname arg1 arg2 ; #传递参数
```

以下是函数`fname`的定义。在函数`fname`中，包含了各种访问函数参数的方法。

```
fname()
{
  echo $1, $2; #访问参数1和参数2
  echo "$@";#以列表的方式一次性打印所有参数
  echo "$*"; #类似于$@，但是参数被作为单个实体
  return 0; #返回值
}
```

类似地，参数可以传递给脚本并通过`script:$0`（脚本名）访问。

- `$1`是第一个参数。
- `$2`是第二个参数。
- `$n`是第*n*个参数。
- `"$@"` 被扩展成 `"$1"` `"$2"` `"$3"`等。
- `"$*"` 被扩展成 `"$1c$2c$3"`，其中c是IFS的第一个字符。
- `"$@"` 要比`"$*"`用得多。由于 `"$*"`将所有的参数当做单个字符串，因此它很少被使用。

1.12.2　补充内容

让我们再研究一些Bash函数的技巧。

1. 递归函数

在Bash中，函数同样支持递归（可以调用自身的函数）。例如，`F() { echo $1; F hello; sleep 1; }`。

> **Fork炸弹**
>
> ```
> :(){ :|:& };:
> ```
>
> 这个递归函数能够调用自身，不断地生成新的进程，最终造成拒绝服务攻击。函数调用前的&将子进程放入后台。这段危险的代码会分支出大量的进程，因而称为Fork炸弹。
>
> 上面这段代码要理解起来可不容易。请参阅维基百科http://en.wikipedia.org/wiki/Fork_bomb，那里列出了有关Fork炸弹的细节以及更详细的解释。
>
> 可以通过修改配置文件/etc/security/limits.conf来限制可生成的最大进程数来避开这枚炸弹。

2. 导出函数

函数也能像环境变量一样用export导出，如此一来，函数的作用域就可以扩展到子进程中，例如：

```
export -f fname
```

3. 读取命令返回值（状态）

我们可以按照下面的方式获取命令或函数的返回值：

```
cmd;
echo $?;
```

$? 会给出命令cmd的返回值。

返回值被称为**退出状态**。它可用于分析命令执行成功与否。如果命令成功退出，那么退出状态为0，否则为非0。

我们可以按照下面的方法检测某个命令是否成功结束：

```
#!/bin/bash
#文件名: success_test.sh
CMD="command" #command指代你要检测退出状态的目标命令
$CMD
if [ $? -eq 0 ];
then
    echo "$CMD executed successfully"
else
    echo "$CMD terminated unsuccessfully"
fi
```

4. 向命令传递参数

命令的参数能够以不同的格式进行传递。假设-p、-v是可用选项，-k N是另一个可以接受数字的选项，同时该命令还接受一个文件名作为参数，那么，它有如下几种执行方式：

- $ command -p -v -k 1 file
- $ command -pv -k 1 file
- $ command -vpk 1 file
- $ command file -pvk 1

1.13　将命令序列的输出读入变量

shell脚本最棒的特性之一就是可以轻松地将多个命令或工具组合起来生成输出。一个命令的输出可以作为另一个命令的输入，而这个命令的输出又会传递至另一个命令，依次类推。这种命令组合的输出可以被存储在一个变量中。这则攻略将演示如何组合多个命令以及如何读取其输出。

1.13.1　预备知识

输入通常是通过stdin或参数传递给命令。输出要么出现在stderr，要么出现在stdout。当我们组合多个命令时，通常将stdin用于输入，stdout用于输出。

此时，这些命令被称为**过滤器**（filter）。我们使用管道（pipe）连接每个过滤器。管道操作符是|。例如：

```
$ cmd1 | cmd2 | cmd3
```

这里我们组合了3个命令。cmd1的输出传递给cmd2，而cmd2的输出传递给cmd3，最终的输出（来自cmd3）将会被打印或导入某个文件。

1.13.2　实战演练

我们通常使用管道并利用子shell的方式将多个文件的输出组合起来。方法如下所示。

(1) 先从组合两个命令开始：

```
$ ls | cat -n > out.txt
```

ls的输出（当前目录内容的列表）被传给cat -n，后者将通过stdin所接收到输入内容加上行号，然后将输出重定向到文件out.txt。

(2) 我们可以用下面的方法读取由管道相连的命令序列的输出：

```
cmd_output=$(COMMANDS)
```

这种方法被称为**子shell**。例如：

```
cmd_output=$(ls | cat -n)
echo $cmd_output
```

另一种被称为**反引用**（有些人们也称它为**反标记**）的方法也可以用于存储命令输出：

```
cmd_output=`COMMANDS`
```

例如：

```
cmd_output=`ls | cat -n`
echo $cmd_output
```

反引用与单引号可不是一回事，它位于键盘的～键上。

1.13.3　补充内容

有很多种方法可以给命令分组。来看看其中的几种。

1. 利用子shell生成一个独立的进程

子shell本身就是独立的进程。可以使用()操作符来定义一个子shell：

```
pwd;
(cd /bin; ls);
pwd;
```

当命令在子shell中执行时，不会对当前shell有任何影响；所有的改变仅限于子shell内。例如，当用cd命令改变子shell的当前目录时，这种变化不会反映到主shell环境中。

pwd命令打印出工作目录的路径。

cd命令将当前目录更改为给定的目录路径。

2. 通过引用子shell的方式保留空格和换行符

假设我们使用子shell或反引用的方法将命令的输出读入一个变量中，可以将它放入双引号中，以保留空格和换行符（\n）。例如：

```
$ cat text.txt
1
2
3

$ out=$(cat text.txt)
$ echo $out
1 2 3 # Lost \n spacing in 1,2,3

$ out="$(cat tex.txt)"
$ echo$out
1
2
3
```

1.14　不使用回车键来读取 *n* 个字符

read是一个重要的Bash命令，它用于从键盘或标准输入中读取文本。我们可以使用read以交互的形式读取来自用户的输入，不过read能做的可远不止这些。任何编程语言的输入库大多都是从键盘读取输入；但只有当回车键按下的时候，才标志着输入完毕。在有些重要情形下是没法按回车键的，输入结束与否是基于字符数或某个特定字符来决定的。例如，在一个游戏中，当按下 + 键时，小球就会向上移动。那么若每次都要按下 + 键，然后再按回车键来确认已经按过 + 键，这就显然太低效了。read命令提供了一种不需要按回车键就能够搞定这个任务的方法。

实战演练

你可以借助read命令的各种选项来实现不同的效果。方法如下所示。

(1) 下面的语句从输入中读取*n*个字符并存入变量variable_name：

```
read -n number_of_chars variable_name
```

例如：

```
$ read -n 2 var
$ echo $var
```

(2) 用无回显的方式读取密码：

```
read -s var
```

(3) 显示提示信息：

```
read -p "Enter input:"  var
```

(4) 在特定时限内读取输入：

```
read -t timeout var
```

例如：

```
$ read -t 2 var
#在2秒内将键入的字符串读入变量var
```

(5) 用特定的定界符作为输入行的结束：

```
read -d delim_char var
```

例如：

```
$ read -d ":" var
hello:#var 被设置为 hello
```

1.15　运行命令直至执行成功

在日常工作中使用shell时，有时候命令只有满足某些条件或是某种外部事件（例如文件可以被下载）操作才能够成功执行。这种情况下，你可能希望重复执行命令，直到成功为止。

1.15.1　实战演练

按照以下方式定义函数：

```
repeat()
{
  while true
  do
    $@ && return
  done
}
```

或者把它放入shell的rc文件，更便于使用：

```
repeat() { while true; do $@ && return; done }
```

1.15.2　工作原理

我们创建了函数repeat，它包含了一个无限while循环，该循环执行以参数形式（通过$@访问）传入函数的命令。如果命令执行成功，则返回，进而退出循环。

1.15.3　补充内容

我们已经看过了用于重复执行命令，直到其执行成功的基本做法。接着来看看更高效的方式。

1. 一种更快的做法

在大多数现代系统中，true是作为/bin中的一个二进制文件来实现的。这就意味着每执行一次while循环，shell就不得不生成一个进程。如果不想这样，可以使用shell内建的“:”命令，它总是会返回为0的退出码：

```
repeat() { while :; do $@ && return; done }
```

尽管可读性不高，但是肯定比第一种方法快。

2. 增加延时

假设你要用repeat()从Internet上下载一个暂时不可用的文件，不过这个文件只需要等一会就能下载。方法如下：

```
repeat wget -c http://www.example.com/software-0.1.tar.gz
```

如果采用这种形式，需要向www.example.com发送很多数据，有可能会对服务器产生影响。（甚至也会给你造成麻烦，如果服务器认为你是在发送垃圾信息，就可能把你列入黑名单。）要解决这个问题，我们可以修改函数，加入一段短暂的延时：

```
repeat() { while :; do $@ && return; sleep 30; done }
```

这使得命令每30秒运行一次。

1.16　字段分隔符和迭代器

内部字段分隔符（Internal Field Separator，IFS）是shell脚本编程中的一个重要概念。在处理文本数据时，它的用途可不小。我们将会讨论把单个数据流划分成不同数据元素的定界符（delimiter）。内部字段分隔符是用于特定用途的定界符。IFS是存储定界符的环境变量。它是当前shell环境使用的默认定界字符串。

考虑一种情形：我们需要迭代一个字符串或**逗号分隔型数值**（Comma Separated Value，CSV）中的单词。如果是前一种，则使用IFS="."；如果是后一种，则使用IFS=","。接下来看看具体的做法。

1.16.1　预备知识

考虑CSV数据的情况：

```
data="name,sex,rollno,location"
```

我们可以使用IFS读取变量中的每一个条目。

```
oldIFS=$IFS
IFS=, now,
for item in $data;
do
    echo Item: $item
done

IFS=$oldIFS
```

输出如下：

```
Item: name
Item: sex
Item: rollno
Item: location
```

IFS的默认值为空白字符（换行符、制表符或者空格）。

当IFS被设置为逗号时，shell将逗号视为一个定界符，因此变量`$item`在每次迭代中读取由逗号分隔的子串作为变量值。

如果没有把IFS设置成逗号，那么上面的脚本会将全部数据作为单个字符串打印出来。

1.16.2　实战演练

让我们以/etc/passwd为例，看看IFS的另一种用法。在文件/etc/passwd中，每一行包含了由冒号分隔的多个条目。文件中的每行都对应一位用户的相关属性。

考虑这样的输入：`root:x:0:0:root:/root:/bin/bash`。每行的最后一项指定了用户的默认shell。可以按照下面的方法巧妙地利用IFS打印出用户以及他们默认的shell：

```
#!/bin/bash
#用途：演示IFS的用法
line="root:x:0:0:root:/root:/bin/bash"
oldIFS=$IFS;
IFS=":"
count=0
for item in $line;
do
```

```
    [ $count -eq 0 ]  && user=$item;
    [ $count -eq 6 ]  && shell=$item;
   let count++
done;
IFS=$oldIFS
echo $user\'s shell is $shell;
```

输出为：

```
root's shell is /bin/bash
```

对一系列值进行迭代时，循环非常有用。Bash提供了多种类型的循环。下面就来看看怎么样使用它们。

- for循环

```
for var in list;
do
    commands; #使用变量$var
done
```

list可以是一个字符串，也可以是一个序列。

我们可以轻松地生成不同的序列。

echo {1..50}能够生成一个从1~50的数字列表。echo {a..z}或{A..Z}或{a..h}可以生成字母列表。同样，我们可以将这些方法结合起来对数据进行拼接（concatenate）。下面的代码中，变量i在每次迭代的过程里都会保存一个字符，范围从a~z：

```
for i in {a..z}; do actions; done;
```

for循环也可以采用C语言中for循环的格式。例如：

```
for((i=0;i<10;i++))
{
    commands; #使用变量$i
}
```

- while循环

```
while condition
do
    commands;
done
```

用true作为循环条件能够产生无限循环。

- until循环

在Bash中还可以使用一个特殊的循环until。它会一直执行循环，直到给定的条件为真。

例如：

```
x=0;
until [ $x -eq 9 ]; #条件是[$x -eq 9 ]
do
    let x++; echo $x;
done
```

1.17　比较与测试

程序中的流程控制是由比较语句和测试语句处理的。Bash同样具备多种与Unix系统级特性相兼容的执行测试的方法。我们可以用if、if else以及逻辑运算符进行测试，用比较运算符来比较数据项。除此之外，还有一个test命令也可以用于测试。这些命令的用法如下。

实战演练

来看看用于比较和测试的各种方法：

- if条件

```
if condition;
then
    commands;
fi
```

- else if和else

```
if condition;
then
    commands;
else if condition; then
    commands;
else
    commands;
fi
```

> if和else语句可以进行嵌套。if的条件判断部分可能会变得很长，但可以用逻辑运算符将它变得简洁一些：
>
> - [condition] && action; # 如果condition为真，则执行action；
> - [condition] || action; # 如果condition为假，则执行action。
>
> &&是逻辑与运算符，||是逻辑或运算符。编写Bash脚本时，这是一个很有用的技巧。现在来了解一下条件和比较操作。

❑ 算术比较

条件通常被放置在封闭的中括号内。一定要注意在[或]与操作数之间有一个空格。如果忘记了这个空格，脚本就会报错。例如：

```
[$var -eq 0 ] or [ $var -eq 0]
```

对变量或值进行算术条件判断：

```
[ $var -eq 0 ]  #当 $var 等于 0 时，返回真
[ $var -ne 0 ] #当 $var 为非 0 时，返回真
```

其他重要的操作符如下所示。

- -gt：大于。
- -lt：小于。
- -ge：大于或等于。
- -le：小于或等于。

可以按照下面的方法结合多个条件进行测试：

```
[ $var1 -ne 0 -a $var2 -gt 2 ]  #使用逻辑与-a
[ $var1 -ne 0 -o var2 -gt 2 ] #逻辑或 -o
```

❑ 文件系统相关测试

我们可以使用不同的条件标志测试不同的文件系统相关的属性。

- [-f $file_var]：如果给定的变量包含正常的文件路径或文件名，则返回真。
- [-x $var]：如果给定的变量包含的文件可执行，则返回真。
- [-d $var]：如果给定的变量包含的是目录，则返回真。
- [-e $var]：如果给定的变量包含的文件存在，则返回真。
- [-c $var]：如果给定的变量包含的是一个字符设备文件的路径，则返回真。
- [-b $var]：如果给定的变量包含的是一个块设备文件的路径，则返回真。
- [-w $var]：如果给定的变量包含的文件可写，则返回真。
- [-r $var]：如果给定的变量包含的文件可读，则返回真。
- [-L $var]：如果给定的变量包含的是一个符号链接，则返回真。

使用方法如下：

```
fpath="/etc/passwd"
if [ -e $fpath ]; then
    echo File exists;
else
    echo Does not exist;
fi
```

- 字符串比较

使用字符串比较时，最好用双中括号，因为有时候采用单个中括号会产生错误，所以最好避开它们。

可以用下面的方法检查两个字符串，看看它们是否相同。

- `[[ $str1 = $str2 ]]`：当str1等于str2时，返回真。也就是说，str1和str2包含的文本是一模一样的。
- `[[ $str1 == $str2 ]]`：这是检查字符串是否相等的另一种写法。

也可以检查两个字符串是否不同。

- `[[ $str1 != $str2 ]]`：如果str1和str2不相同，则返回真。

我们还可以检查字符串的字母序情况，具体如下所示。

- `[[ $str1 > $str2 ]]`：如果str1的字母序比str2大，则返回真。
- `[[ $str1 < $str2 ]]`：如果str1的字母序比str2小，则返回真。
- `[[ -z $str1 ]]`：如果str1包含的是空字符串，则返回真。
- `[[ -n $str1 ]]`：如果str1包含的是非空字符串，则返回真。

> 注意在=前后各有一个空格。如果忘记加空格，那就不是比较关系了，而变成了赋值语句。

使用逻辑运算符 && 和 || 能够很容易地将多个条件组合起来：

```
if [[ -n $str1 ]] && [[ -z $str2 ]] ;
then
    commands;
fi
```

例如：

```
str1="Not empty "
str2=""
if [[ -n $str1 ]] && [[ -z $str2 ]];
then
    echo str1 is nonempty and str2 is empty string.
fi
```

输出如下：

```
str1 is nonempty and str2 is empty string.
```

test命令可以用来执行条件检测。用test可以避免使用过多的括号。之前讲过的[]中的测试条件同样可以用于test命令。

例如：

```
if  [ $var -eq 0 ]; then echo "True"; fi
```

也可以写成：

```
if  test $var -eq 0 ; then echo "True"; fi
```

第 2 章

命令之乐

本章内容

- ❑ 用cat进行拼接
- ❑ 录制与回放终端会话
- ❑ 文件查找与文件列表
- ❑ 玩转xargs
- ❑ 用tr进行转换
- ❑ 校验和与核实
- ❑ 加密工具与散列
- ❑ 排序、单一与重复
- ❑ 临时文件命名与随机数
- ❑ 分割文件和数据
- ❑ 根据扩展名切分文件名
- ❑ 用rename和mv批量重命名文件
- ❑ 拼写检查与词典操作
- ❑ 交互输入自动化
- ❑ 利用并行进程加速命令执行

2.1 简介

类Unix系统享有最棒的命令行工具。它们帮助我们完成各类任务，使我们的工作变得更轻松。尽管这些命令各有侧重，在实践中你可以通过结合多个命令来解决复杂的问题。一些经常用到的命令是grep、awk、sed和find。

掌握Unix/Linux命令行可谓是一门艺术。实践得越多，收益就越大。本章将为你介绍一些最有趣同时也是最实用的命令。

2.2 用 cat 进行拼接

cat是命令行玩家首先必须学习的命令之一。它通常用于读取、显示或拼接文件内容，不过cat的能力远不止如此。用一行命令将来自标准输入以及文件的数据给组合起来，这可是个让人挠头的难题。通常的做法是将stdin重定向到一个文件，然后再将两个文件组合到一起。不过我们可以使用cat命令一次性搞定。接下来你会看到cat的基本用法和高级用法。

2.2.1 实战演练

cat命令是一个平日里经常会使用到的简单命令。它本身表示concatenate（拼接）。

用cat读取文件内容的一般写法是：

```
$ cat file1 file2 file3 ...
```

这个命令将作为命令行参数的文件内容拼接在一起。

- 打印单个文件的内容：

```
$ cat file.txt
This is a line inside file.txt
This is the second line inside file.txt
```

- 打印多个文件的内容：

```
$ cat one.txt two.txt
This is line from one.txt
This is line from two.txt
```

2.2.2 工作原理

cat的用法多种多样。让我们来看看其中的一些。

cat命令不仅可以读取文件、拼接数据，还能够从标准输入中进行读取。

从标准输入中读取需要使用管道操作符：

```
OUTPUT_FROM_SOME COMMANDS | cat
```

类似地，我们可以用cat将来自输入文件的内容与标准输入拼接在一起，将stdin和另一个文件中的数据结合起来。方法如下：

```
$ echo 'Text through stdin' | cat - file.txt
```

在上面的代码中，-被作为stdin文本的文件名。

2.2.3 补充内容

cat命令还有一些用于文件查看的选项。来看看它们的用法。

1. 摆脱多余的空白行

有时候文本文件中可能包含多处连续的空白行。如果你需要删除这些额外的空白行，使用下面的方法：

```
$ cat -s file
```

例如:

```
$ cat multi_blanks.txt
line 1



line2

line3


line4

$ cat -s multi_blanks.txt #压缩相邻的空白行
line 1

line2

line3

line4
```

另外，也可以用tr删除所有的z空白行，我们会在2.6节详细讨论。

2. 将制表符显示为^|

单从视觉上很难将制表符同连续的空格区分开。而在用Python编写程序时，用于代码缩进的制表符以及空格是具有特殊含义的。因此，若在应该使用空格的地方误用了制表符的话，就会产生缩进错误。仅仅在文本编辑器中进行观察是很难发现这种错误的。cat有一个特性，可以将制表符着重标记出来。该特性对排除缩进错误非常有用。用cat命令的 -T选项能够将制表符标记成 ^|。例如:

```
$ cat file.py
def function():
    var = 5
        next = 6
    third = 7

$ cat -T file.py
def function():
^Ivar = 5
        next = 6
^Ithird = 7^I
```

3. 行号

使用cat命令的-n选项会在输出的每一行内容之前加上行号。别担心，cat命令绝不会修改你的文件，它只是根据用户提供的选项在stdout中生成一个修改过的输出而已。例如：

```
$ cat lines.txt
line
line
line

$ cat -n lines.txt
     1 line
     2 line
     3 line
```

> -n甚至会为空白行加上行号。如果你想跳过空白行，那么可以使用选项-b。

2.3 录制并回放终端会话

当你需要为别人在终端上演示某些操作或是需要准备一个命令行教程时，通常得一边手动输入命令一边演示，或者也可以录制一段屏幕演示视频，然后再回放出来。其实也有其他的实现方法。利用script和scriptreplay命令，我们可以录制命令的次序以及时序，将相关数据记录在文本文件中。利用这些文件，其他人可以在终端上回放并查看命令的输出。

2.3.1 预备知识

script和scirptreplay命令在绝大多数GNU/Linux发行版上都可以找到。把终端会话记录到一个文件中挺好玩的。你可以通过录制终端会话来制作命令行教学视频教程，也可以与他人分享会话记录文件，共同研究如何使用命令行完成某项任务。

2.3.2 实战演练

开始录制终端会话：

```
$ script -t 2> timing.log -a output.session
type commands;
…
..
exit
```

> 注意，对于不支持单独将stderr重定向到文件的shell（比如csh shell），这则攻略是没法使用的。

两个配置文件被当做script命令的参数。其中一个文件（timing.log）用于存储时序信息，描述每一个命令在何时运行；另一个文件（output.session）用于存储命令输出。-t选项用于将时序数据导入stderr。2>则用于将stderr重定向到timing.log。

利用这两个文件：timing.log（存储时序信息）和output.session（存储命令输出信息），我们可以按照下面的方法回放命令执行过程：

```
$ scriptreplay timing.log output.session
# 按播放命令序列输出
```

2.3.3　工作原理

通常，我们会录制桌面环境视频来作为教程使用。但是视频需要大量的存储空间，而终端脚本文件仅仅是一个文本文件，其文件大小不过是KB级别。

你可以把timing.log和output.session文件分享给任何想在自己的终端上回放这段终端会话的人。

2.4　文件查找与文件列表

find是Unix/Linux命令行工具箱中最棒的工具之一。该命令对于编写shell脚本所起到的功用不可小视，但是多数人却无法最大程度发挥它的功效。这则攻略讨论了find的大多数常见用法。

2.4.1　预备知识

find命令的工作方式如下：沿着文件层次结构向下遍历，匹配符合条件的文件，执行相应的操作。下面来看看find命令的各种应用场景和基本用法。

2.4.2　实战演练

要列出当前目录及子目录下所有的文件和文件夹，可以采用下面的写法：

```
$ find base_path
```

bash_path可以放在任意位置（例如 /home/slynux），find会从该位置开始向下查找。

例如：

```
$ find . -print
# 打印文件和目录的列表
```

. 指定当前目录，.. 指定父目录。这是Unix文件系统中的约定用法。

-print指明打印出匹配文件的文件名（路径）。当使用 -print时，'\n'作为用于对输出的文件名进行分隔。就算你忽略-print，find命令仍会打印出文件名。

-print0指明使用'\0'作为匹配的文件名之间的定界符。当文件名中包含换行符时，这个方法就有用武之地了。

2.4.3 补充内容

至此，我们已经学习了find命令最常见的用法。作为一款强大的命令行工具，find命令包含了诸多值得一提的选项。接下来看看find命令的其他功能。

1. 根据文件名或正则表达式进行搜索

选项-name的参数指定了文件名所必须匹配的字符串。我们可以将通配符作为参数使用。*.txt能够匹配所有以.txt结尾的文件名。选项 -print在终端中打印出符合条件（例如 -name）的文件名或文件路径，这些匹配条件通过find命令的选项给出。

```
$ find /home/slynux -name "*.txt" -print
```

find命令有一个选项 -iname（忽略字母大小写），该选项的作用和 -name类似，只不过在匹配名字时会忽略大小写。

例如：

```
$ ls
example.txt  EXAMPLE.txt  file.txt
$ find . -iname "example*" -print
./example.txt
./EXAMPLE.txt
```

如果想匹配多个条件中的一个，可以采用OR条件操作：

```
$ ls
new.txt  some.jpg  text.pdf
$ find . \( -name "*.txt" -o -name "*.pdf" \) -print
./text.pdf
./new.txt
```

上面的代码会打印出所有的.txt和.pdf文件，是因为这个find命令能够匹配所有这两类文件。\(以及\)用于将 -name "*.txt" -o -name "*.pdf"视为一个整体。

选项-path的参数可以使用通配符来匹配文件路径。-name总是用给定的文件名进行匹配。-path则将文件路径作为一个整体进行匹配。例如：

```
$ find /home/users -path "*/slynux/*" -print
```

这会匹配以下路径：

```
/home/users/list/slynux.txt
/home/users/slynux/eg.css
```

> 选项-regex的参数和-path的类似，只不过-regex是基于正则表达式来匹配文件路径的。

正则表达式是通配符匹配的高级形式，它可以指定文本模式。我们借助这种模式来匹配文本及进行打印。使用正则表达式进行文本匹配的一个典型例子就是从一堆文本中解析出所有的E-mail地址。E-mail地址通常采用name@host.root这种形式，所以可以将其一般化为[a-z0-9]+@[a-z0-9]+.[a-z0-9]+。符号+ 指明在它之前的字符类中的字符可以出现一次或多次。

下面的命令匹配.py或.sh文件：

```
$ ls
new.PY  next.jpg  test.py
$ find . -regex ".*\(\.py\|\.sh\)$"
./test.py
```

类似地，-iregex可以让正则表达式忽略大小写。例如：

```
$ find . -iregex ".*\(\.py\|\.sh\)$"
./test.py
./new.PY
```

> 我们会在第4章学习到更多有关正则表达式的内容。

2. 否定参数

find也可以用“!”否定参数的含义。例如：

```
$ find . ! -name "*.txt" -print
```

上面的find命令能够匹配所有不以.txt结尾的文件名。下面就是这个命令的运行结果：

```
$ ls
list.txt  new.PY  new.txt  next.jpg  test.py

$ find . ! -name "*.txt" -print
.
./next.jpg
```

```
./test.py
./new.PY
```

3. 基于目录深度的搜索

find命令在使用时会遍历所有的子目录。我们可以采用深度选项-maxdepth和 -mindepth来限制find命令遍历的目录深度。

大多数情况下，我们只需要在当前目录中进行搜索，无须再继续向下查找。对于这种情况，我们使用深度选项来限制find命令向下查找的深度。如果只允许find在当前目录中查找，深度可以设置为1；当需要向下两级时，深度可以设置为2；其他情况可以依次类推。

可以用-maxzdepth指定最大深度。与此相似，我们也可以指定一个最小的深度，告诉find应该从此处开始向下查找。如果我们想从第二级目录开始搜索，那么就使用 -mindepth设置最小深度。使用下列命令将find命令向下的最大深度限制为1：

```
$ find . -maxdepth 1 -name "f*" -print
```

该命令列出当前目录下的所有文件名以f打头的文件。即使有子目录，也不会被打印或遍历。与之类似，-maxdepth 2最多向下遍历两级子目录。

-mindepth类似于 -maxdepth，不过它设置的是find开始遍历的最小深度。这个选项可以用来查找并打印那些距离起始路径一定深度的所有文件。例如，打印出深度距离当前目录至少两个子目录的所有文件：

```
$ find . -mindepth 2 -name "f*" -print
./dir1/dir2/file1
./dir3/dir4/f2
```

即使当前目录或dir1和dir3中包含有文件，它们也不会被打印出来。

> -maxdepth和-mindepth应该作为find的第三个参数出现。如果作为第4个或之后的参数，就可能会影响到find的效率，因为它不得不进行一些不必要的检查。例如，如果-maxdepth作为第四个参数，-type作为第三个参数，find首先会找出符合-type的所有文件，然后在所有匹配的文件中再找出符合指定深度的那些。但是如果反过来，目录深度作为第三个参数，-type作为第四个参数，那么find就能够在找到所有符合指定深度的文件后，再检查这些文件的类型，这才是最有效的搜索之道。

4. 根据文件类型搜索

Unix类系统将一切都视为文件。文件具有不同的类型，例如普通文件、目录、字符设备、块设备、符号链接、硬链接、套接字以及FIFO等。

-type可以对文件搜索进行过滤。借助这个选项，我们可以为find命令指明特定的文件匹配类型。

只列出所有的目录：

```
$ find . -type d -print
```

将文件和目录分别列出可不是个容易事。不过有了find就好办了。例如，只列出普通文件：

```
$ find . -type f -print
```

只列出符号链接：

```
$ find . -type l -print
```

你可以按照表2-1列出的内容用type参数来匹配所需要的文件类型。

表　2-1

文件类型	类型参数
普通文件	f
符号链接	l
目录	d
字符设备	c
块设备	b
套接字	s
FIFO	p

5. 根据文件时间进行搜索

Unix/Linux文件系统中的每一个文件都有三种时间戳，如下所示。

- **访问时间**（-atime）：用户最近一次访问文件的时间。
- **修改时间**（-mtime）：文件内容最后一次被修改的时间。
- **变化时间**（-ctime）：文件元数据（例如权限或所有权）最后一次改变的时间。

> 在Unix中并没有所谓“创建时间”的概念。

-atime、-mtime、-ctime可作为find的时间选项。它们可以用整数值指定，单位是天。这些整数值通常还带有 - 或 + ：- 表示小于，+ 表示大于。

- 打印出在最近7天内被访问过的所有文件：

```
$ find . -type f -atime -7 -print
```

- 打印出恰好在7天前被访问过的所有文件：

```
$ find . -type f -atime 7 -print
```

- 打印出访问时间超过7天的所有文件：

```
$ find . -type f -atime +7 -print
```


类似地，我们可以根据修改时间，用-mtime进行搜索，也可以根据变化时间，用-ctime进行搜索。

-atime、-mtime以及-ctime都是基于时间的参数，其计量单位是“天”。还有其他一些基于时间的参数是以分钟作为计量单位的。这些参数包括：

- -amin（访问时间）；
- -mmin（修改时间）；
- -cmin（变化时间）。

举例如下。

打印出访问时间超过7分钟的所有文件：

```
$ find . -type f -amin +7 -print
```

find另一个漂亮的特性是 -newer参数。使用 -newer，我们可以指定一个用于比较时间戳的参考文件，然后找出比参考文件更新的（更近的修改时间）所有文件。

例如，找出比file.txt修改时间更近的所有文件：

```
$ find . -type f -newer file.txt -print
```

find命令的时间戳处理选项对编写系统备份和维护脚本很有帮助。

6. 基于文件大小的搜索

根据文件的大小，可以这样搜索：

```
$ find . -type f -size +2k
# 大于2KB的文件

$ find . -type f -size -2k
# 小于2KB的文件

$ find . -type f -size 2k
# 大小等于2KB的文件
```

除了k之外，还可以用其他文件大小单元。

- b —— 块（512字节）。
- c —— 字节。
- w —— 字（2字节）。
- k —— 1024字节。
- M —— 1024K字节。
- G —— 1024M字节。

7. 删除匹配的文件

-delete可以用来删除find查找到的匹配文件。

删除当前目录下所有的 .swp文件：

```
$ find . -type f -name "*.swp" -delete
```

8. 基于文件权限和所有权的匹配

也可以根据文件权限进行文件匹配。列出具有特定权限的所有文件：

```
$ find . -type f -perm 644 -print
# 打印出权限为644的文件
```

-perm指明find应该只匹配具有特定权限值的文件。文件权限会在3.5节进行讲解。

以Apache Web服务器为例。Web服务器上的PHP文件需要具有合适的执行权限。我们可以用下面的方法找出那些没有设置好执行权限的PHP文件：

```
$ find . -type f -name "*.php" ! -perm 644 -print
```

也可以根据文件的所有权进行搜索。用选项 -user USER就能够找出由某个特定用户所拥有的文件。

参数USER可以是用户名或UID。

例如，打印出用户slynux拥有的所有文件：

```
$ find . -type f -user slynux -print
```

9. 利用find执行命令或动作

find命令可以借助选项-exec与其他命名进行结合。-exec算得上是find最强大的特性之一。

在前一节中，我们用 -perm找出了所有权限不当的PHP文件。这次的任务也差不多，我们需要将某位用户（比如root）全部文件的所有权更改成另一位用户（比如Web服务器默认的Apache用户www-data），那么就可以用 -user找出root拥有的所有文件，然后用-exec更改所有权。

> 你必须以root用户的身份执行find命令才能够进行所有权的更改。

示例如下：

```
# find . -type f -user root -exec chown slynux {} \;
```

在这个命令中，{}是一个与 -exec选项搭配使用的特殊字符串。对于每一个匹配的文件，{}会被替换成相应的文件名。例如，find命令找到两个文件test1.txt和test2.txt，其所有者均为slynux，那么find就会执行：

```
chown slynux {}
```

它会被解析为chown slynux test1.txt和chown slynux test2.txt。

> 有时候我们并不希望对每个文件都执行一次命令。我们更希望使用文件列表作为命令参数，这样就可以少运行几次命令了。如果是这样，可以在exec中使用+来代替;。

另一个例子是将给定目录中的所有C程序文件拼接起来写入单个文件all_c_files.txt。我们可以用find找到所有的C文件，然后结合 -exec使用cat命令：

```
$ find . -type f -name "*.c" -exec cat {} \;>all_c_files.txt
```

-exec之后可以接任何命令。{}表示一个匹配。对于任何匹配的文件名，{}均会被该文件名所替换。

我们使用 > 操作符将来自find的数据重定向到all_c_files.txt文件，没有使用>>（追加）的原因是因为find命令的全部输出就只有一个数据流（stdin），而只有当多个数据流被追加到单个文件中时才有必要使用>>。

例如，用下列命令将10天前的 .txt文件复制到OLD目录中：

```
$ find . -type f -mtime +10 -name "*.txt" -exec cp {} OLD  \;
```

find命令同样可以采用类似的方法与其他命令结合起来。

> **-exec结合多个命令**
>
> 我们无法在-exec参数中直接使用多个命令。它只能够接受单个命令，不过我们可以耍一个小花招。把多个命令写到一个shell脚本中（例如command.sh），然后在-exec中使用这个脚本：
>
> ```
> -exec ./commands.sh {} \;
> ```

-exec能够同printf结合来生成有用的输出信息。例如：

```
$ find . -type f -name "*.txt" -exec printf "Text file: %s\n" {} \;
```

10. 让find跳过特定的目录

在搜索目录并执行某些操作时，有时为了提高性能，需要跳过一些子目录。例如，程序员会在版本控制系统（如Git）管理的开发源码树中查找特定的文件，源代码层级结构中的每个子目录里都会包含一个.git目录。（.git存储每个目录相关的版本控制信息。）因为与版本控制相关的目录对我们而言并没有什么用处，所以没必要去搜索这些目录。将某些文件或目录从搜索过程中排除在外的技巧被称为**修剪**，其操作方法如下：

```
$ find devel/source_path  \( -name ".git" -prune \) -o \( -type f -print \)

# Instead of \( -type -print \),而是选择需要的过滤器
```

以上命令打印出不包括在.git目录中的所有文件的名称（路径）。

\(-name ".git" -prune \)的作用是用于进行排除，它指明了 .git目录应该被排除在外，而\(-type f -print \)指明了需要执行的动作。这些动作需要被放置在第二个语句块中（打印出所有文件的名称和路径）。

2.5 玩转 xargs

我们可以用管道将一个命令的stdout（标准输出）重定向到另一个命令的stdin（标准输入）。例如：

```
cat foo.txt | grep "test"
```

但是，有些命令只能以命令行参数的形式接受数据，而无法通过stdin接受数据流。在这种情况下，我们没法用管道来提供那些只有通过命令行参数才能提供的数据。

那就只能另辟蹊径了。该xargs命令出场了，它擅长将标准输入数据转换成命令行参数。xargs能够处理stdin并将其转换为特定命令的命令行参数。xargs也可以将单行或多行文本输入转换成其他格式，例如单行变多行或是多行变单行。

Bash用户都爱单行命令。单行命令是一个命令序列，各命令之间不使用分号，而是使用管道操作符进行连接。精心编写的单行命令可以更高效、更简捷地完成任务。就文本处理而言，需要具备扎实的理论知识和实践经验才能够写出适合的单行命令解决方案。xargs就是构建单行命令的重要组件之一。

2.5.1 预备知识

xargs命令应该紧跟在管道操作符之后，以标准输入作为主要的源数据流。它使用stdin并通过提供命令行参数来执行其他命令。例如：

```
command | xargs
```

2.5.2 实战演练

xargs命令把从stdin接收到的数据重新格式化，再将其作为参数提供给其他命令。

xargs可以作为一种替代，其作用类似于find命令中的-exec。下面是各种xargs命令的使用技巧。

- 将多行输入转换成单行输出。

  只需要将换行符移除，再用" "（空格）进行代替，就可以实现多行输入的转换。'\n'被解释成一个换行符，换行符其实就是多行文本之间的定界符。利用xargs，我们可以用空格替换掉换行符，这样就能够将多行文本转换成单行文本：

```
$ cat example.txt # 样例文件
1 2 3 4 5 6
7 8 9 10
11 12

$ cat example.txt | xargs
1 2 3 4 5 6 7 8 9 10 11 12
```

- 将单行输入转换成多行输出。

  指定每行最大的参数数量*n*，我们可以将任何来自stdin的文本划分成多行，每行*n*个参数。每一个参数都是由" "（空格）隔开的字符串。空格是默认的定界符。下面的方法可以将单行划分成多行：

```
$ cat example.txt | xargs -n 3
1 2 3
4 5 6
7 8 9
10 11 12
```

2.5.3 工作原理

xargs命令数量众多的选项使其能够适用于多种问题场景。让我们来看看如何能够巧妙地运用这些选项来解决问题。

可以用自己的定界符来分隔参数。用-d选项为输入指定一个定制的定界符：

```
$ echo "splitXsplitXsplitXsplit" | xargs -d X
split split split split
```

2

在上面的代码中，stdin是一个包含了多个X字符的字符串。我们可以用 -d将X作为输入定界符。在这里，我们明确指定X作为输入定界符，而在默认情况下，xargs采用**内部字段分隔符**（空格）作为输入定界符。

结合-n选项，我们可以将输入划分成多行，而每行包含两个参数：

```
$ echo "splitXsplitXsplitXsplit" | xargs -d X -n 2
split split
split split
```

2.5.4　补充内容

我们已经从上面的例子中学到了如何将stdin格式化成不同的输出形式以作为参数。现在让我们来学习如何将这些参数传递给命令。

1. 读取stdin，将格式化参数传递给命令

编写一个小型的定制版echo来更好地理解用xargs提供命令行参数的方法：

```
#!/bin/bash
#文件名: cecho.sh

echo $*'#'
```

当参数被传递给文件cecho.sh后，它会将这些参数打印出来，并以 # 字符作为结尾。例如：

```
$ ./cecho.sh arg1 arg2
arg1 arg2 #
```

让我们来看下面这个问题。

- 有一个包含着参数列表的文件（每行一个参数）。我需要用两种方法将这些参数传递给一个命令（比如cecho.sh）。第一种方法，需要每次提供一个参数：

  ```
  ./cecho.sh arg1
  ./cecho.sh arg2
  ./cecho.sh arg3
  ```

  或者，每次需要提供两个或三个参数。提供两个参数时，它看起来像这样：

  ```
  ./cecho.sh arg1 arg2
  ./cecho.sh arg3
  ```

- 第二种方法，需要一次性提供所有的命令参数：

  ```
  ./cecho.sh arg1 arg2 arg3
  ```

先别急着往下看，试着运行一下上面的命令，然后仔细观察输出结果。

上面的问题也可以用xargs来解决。我们有一个名为args.txt的参数列表文件，这个文件的内容如下：

```
$ cat args.txt
arg1
arg2
arg3
```


就第一个问题，我们可以将这个命令执行多次，每次使用一个参数：

```
$ cat args.txt | xargs -n 1 ./cecho.sh
arg1 #
arg2 #
arg3 #
```

每次执行需要X个参数的命令时，使用：

```
INPUT | xargs -n X
```

例如：

```
$ cat args.txt | xargs -n 2 ./cecho.sh
arg1 arg2 #
arg3 #
```

就第二个问题，为了在执行命令时一次性提供所有的参数，可以使用：

```
$ cat args.txt | xargs ./cecho.sh
arg1 arg2 arg3 #
```

在上面的例子中，我们直接为特定的命令（例如cecho.sh）提供命令行参数。这些参数都源于args.txt文件。但实际上除了它们外，我们还需要一些固定不变的命令参数。思考下面这种命令格式：

```
./cecho.sh -p arg1 -l
```

在上面的命令执行过程中，arg1是唯一的可变内容，其余部分都保持不变。我们可以从文件（args.txt）中读取参数，并按照下面的方式提供给命令：

```
./cecho.sh -p arg1 -l
./cecho.sh -p arg2 -l
./cecho.sh -p arg3 -l
```

xargs有一个选项-I，可以提供上面这种形式的命令执行序列。我们可以用-I指定替换字符串，这个字符串在xargs扩展时会被替换掉。如果将-I与xargs结合使用，对于每一个参数，命令都会被执行一次。

试试下面的用法：

```
$ cat args.txt | xargs -I {} ./cecho.sh -p {} -l
-p arg1 -l #
-p arg2 -l #
-p arg3 -l #
```

`-I {}`指定了替换字符串。对于每一个命令参数，字符串`{}`都会被从`stdin`读取到的参数替换掉。

> 使用`-I`的时候，命令以循环的方式执行。如果有3个参数，那么命令就会连同`{}`一起被执行3次。在每一次执行中`{}`都会被替换为相应的参数。

2. 结合find使用xargs

`xargs`和`find`算是一对死党。两者结合使用可以让任务变得更轻松。不过人们通常却是以一种错误的组合方式使用它们。例如：

```
$ find . -type f -name "*.txt"  -print | xargs rm -f
```

这样做很危险。有时可能会删除不必要删除的文件。我们没法预测分隔`find`命令输出结果的定界符究竟是什么（`'\n'`或者`' '`）。很多文件名中都可能会包含空格符（`' '`），因此`xargs`很可能会误认为它们是定界符（例如，hell text.txt会被`xargs`误解为hell和text.txt）。

只要我们把`find`的输出作为`xargs`的输入，就必须将`-print0`与`find`结合使用，以字符null（`'\0'`）来分隔输出。

用`find`匹配并列出所有的 .txt文件，然后用`xargs`将这些文件删除：

```
$ find . -type f -name "*.txt" -print0 | xargs -0 rm -f
```

这样就可以删除所有的.txt文件。`xargs -0`将`\0`作为输入定界符。

3. 统计源代码目录中所有C程序文件的行数

统计所有C程序文件的**行数**（Lines of Code，LOC）是大多数程序员都会遇到的任务。完成这项任务的代码如下：

```
$ find source_code_dir_path -type f -name "*.c" -print0 | xargs -0 wc -l
```

> 如果你想获得有关个人源代码更多的统计信息，有个叫做SLOCCount的工具可以派上用场。现代GNU/Linux发行版一般都包含这个软件包，或者你也可以从http://www.dwheeler.com/sloccount/处下载。

4. 结合**stdin**，巧妙运用**while**语句和子shell

xargs只能以有限的几种方式来提供参数，而且它也不能为多组命令提供参数。要执行包含来自标准输入的多个参数的命令，有一种非常灵活的方法。包含while循环的子shell可以用来读取参数，然后通过一种巧妙的方式执行命令：

```
$ cat files.txt  | ( while read arg; do cat $arg; done )
# 等同于cat files.txt | xargs -I {} cat {}
```

在while循环中，可以将cat $arg替换成任意数量的命令，这样我们就可以对同一个参数执行多条命令。也可以不借助管道，将输出传递给其他命令。这个技巧能够适用于各种问题场景。子shell操作符内部的多个命令可作为一个整体来运行。

```
$ cmd0 | ( cmd1;cmd2;cmd3) | cmd4
```

如果cmd1是cd /，那么就会改变子shell工作目录，然而这种改变仅局限于子shell内部。cmd4则完全不知道工作目录发生了变化。

2.6 用 **tr** 进行转换

tr是Unix命令行专家工具箱中一件简约却不失精美的工具。它经常用来编写优美的单行命令，作用不容小视。tr可以对来自标准输入的内容进行字符替换、字符删除以及重复字符压缩。它可以将一组字符变成另一组字符，因而通常也被称为**转换**（translate）命令。在这则攻略中，我们会看到如何使用tr进行基本的集合转换。

2.6.1 预备知识

tr只能通过stdin（标准输入），而无法通过命令行参数来接受输入。它的调用格式如下：

```
tr [options] set1 set2
```

将来自stdin的输入字符从set1映射到set2，然后将输出写入stdout（标准输出）。set1和set2是字符类或字符集。如果两个字符集的长度不相等，那么set2会不断重复其最后一个字符，直到长度与set1相同。如果set2的长度大于set1，那么在set2中超出set1长度的那部分字符则全部被忽略。

2.6.2 实战演练

将输入字符由大写转换成小写，可以使用下面的命令：

```
$ echo "HELLO WHO IS THIS" | tr 'A-Z' 'a-z'
```

'A-Z' 和 'a-z'都是集合。我们可以按照需要追加字符或字符类来构造自己定制的集合。

'ABD-}'、'aA.,'、'a-ce-x'以及'a-c0-9'等均是合法的集合。定义集合也很简单，不需要书写一长串连续的字符序列，只需要使用“起始字符–终止字符”这种格式就行了。这种写法也可以和其他字符或字符类结合使用。如果“起始字符–终止字符”不是一个连续的字符序列，那么它就会被视为包含了3个元素的集合，也就是：起始字符，–，终止字符。你可以使用像'\t'、'\n'这种特殊字符，也可以使用其他ASCII字符。

2.6.3　工作原理

通过在tr中使用集合的概念，我们可以轻松地将字符从一个集合映射到另一个集合中。下面通过一则示例看看如何用tr进行数字加密和解密。

```
$ echo 12345 | tr '0-9' '9876543210'
87654 # 已加密

$ echo 87654 | tr '9876543210' '0-9'
12345 # 已解密
```

再来看另外一个有趣的例子。

ROT13是一个著名的加密算法。在ROT13算法中，文本加密和解密都使用同一个函数。ROT13按照字母表排列顺序执行13个字母的转换。用tr进行ROT13加密：

```
$ echo "tr came, tr saw, tr conquered." | tr 'a-zA-Z' 'n-za-mN-ZA-M'
```

得到输出：

```
ge pnzr, ge fnj, ge pbadhrerq.
```

对加密后的密文再次使用同样的ROT13函数，我们采用：

```
$ echo ge pnzr, ge fnj, ge pbadhrerq. | tr 'a-zA-Z' 'n-za-mN-ZA-M'
```

得到输出：

```
tr came, tr saw, tr conquered.
```

tr还可以用来将制表符转换成空格：

```
$ tr '\t' ' ' < file.txt
```

2.6.4　补充内容

我们已经看到了tr的一些基本转换功能，接下来看看tr还能帮我们实现的其他功能。

1. 用**tr**删除字符

tr有一个选项-d，可以通过指定需要被删除的字符集合，将出现在stdin中的特定字符清除掉：

```
$ cat file.txt | tr -d  '[set1]'
#只使用set1，不使用set2
```

例如：

```
$ echo "Hello 123 world 456" | tr -d '0-9'
Hello world
# 将stdin中的数字删除并打印出来
```

2. 字符集补集

我们可以利用选项-c来使用set1的补集。下面的命令中，set2是可选的：

```
tr -c [set1] [set2]
```

set1的补集意味着这个集合中包含set1中没有的所有字符。

最典型的用法是从输入文本中将不在补集中的所有字符全部删除。例如：

```
$ echo hello 1 char 2 next 4 | tr -d -c '0-9 \n'
 1  2  4
```

在这里，补集中包含了除数字、空格字符和换行符之外的所有字符。因为指定了-d，所以这些字符全部都被删除。

3. 用**tr**压缩字符

tr命令在很多文本处理环境中大有用武之地。多数情况下，连续的重复字符应该压缩成单个字符。经常需要从事的一项任务就是压缩空白字符。

tr的-s选项可以压缩输入中重复的字符，方法如下：

```
$ echo "GNU is       not     UNIX. Recursive   right ?" | tr -s ' '
GNU is not UNIX. Recursive right ?
# tr -s '[set]'
```

让我们用一种巧妙的方式用tr将文件中的数字列表进行相加：

```
$ cat sum.txt
1
2
3
4
```

```
5

$ cat sum.txt | echo $[ $(tr '\n' '+' ) 0 ]
15
```

这一招是如何起效的?

在上面的命令中,tr用来将'\n'替换成'+',因此我们得到了字符串"1+2+3+…5+",但是在字符串的尾部多了一个操作符+。为了抵消这个多出来的操作符,我们再追加一个0。

$[operation]执行算术运算,因此得到下面的字符串:

```
echo $[ 1+2+3+4+5+0 ]
```

如果我们利用循环从文件中读取数字,然后再进行相加,那肯定得用好几行代码。而如今只用一行就搞定了。

还可以像下面这样使用tr,以摈除多余的换行符:

```
$ cat multi_blanks.txt | tr -s '\n'
line1
line2
line3
line4
```

在上例中,tr将多余的'\n'字符合并为单一的'\n'(换行符)。

4. 字符类

tr可以像使用集合一样使用各种不同的字符类,这些字符类如下所示:

- alnum:字母和数字。
- alpha:字母。
- cntrl:控制(非打印)字符。
- digit:数字。
- graph:图形字符。
- lower:小写字母。
- print:可打印字符。
- punct:标点符号。
- space:空白字符。
- upper:大写字母。
- xdigit:十六进制字符。

可以按照下面的方式选择并使用所需的字符类：

```
tr [:class:] [:class:]
```

例如：

```
tr '[:lower:]' '[:upper:]'
```


2.7　校验和与核实

校验和（checksum）程序用来从文件中生成校验和密钥，然后利用这个校验和密钥核实文件的完整性。文件可以通过网络或任何存储介质分发到不同的地点。出于多种原因，数据有可能在传输过程中丢失了若干位，从而导致文件损坏。这种错误通常发生在从Internet上下载文件，通过网络传输文件，或者CD光盘损坏等。

因此，我们需要采用一些测试方法来确定接收到的文件是否存在错误。用于文件完整性测试的特定密钥就称为**校验和**。

我们对原始文件和接收到的文件都进行校验和计算。通过比对两者的校验和，就能够核实接收到的文件是否正确。如果校验和（一个来自发送地的原始文件，另一个来自目的地的接收文件）相等，就意味着我们接收到了正确的文件，否则用户就不得不重新发送文件并再次比对校验和。

校验和对于编写备份脚本或系统维护脚本来说非常重要，因为它们都会涉及通过网络传输文件。通过使用校验和核实，我们就可以识别出那些在网络传输过程中出现损坏的文件，并进行重发。

在这则攻略中，我们会学习如何计算校验和验证数据完整性。

2.7.1　预备知识

最知名且使用最为广泛的校验和技术是md5sum和SHA-1。它们对文件内容使用相应的算法来生成校验和。下面就来看看如何从文件中生成校验和并核实文件的完整性。

2.7.2　实战演练

为了计算md5sum，使用下列命令：

```
$ md5sum filename
68b329da9893e34099c7d8ad5cb9c940 filename
```

如上所示，`md5sum`是一个32个字符的十六进制串。

将输出的校验和重定向到一个文件，然后用这个MD5文件核实数据的完整性：

```
$ md5sum filename > file_sum.md5
```

2.7.3　工作原理

md5sum校验和计算的方法如下：

```
$ md5sum file1 file2 file3 ..
```

当使用多个文件时，输出中会在每行中包含单个文件的校验和：

```
[checksum1]   file1
[checksum1]   file2
[checksum1]   file3
```

可以按照下面的方法用生成的文件核实数据完整性：

```
$ md5sum -c file_sum.md5
# 这个命令会输出校验和是否匹配的消息
```

如果需要用所有的.md5信息来检查所有的文件，可以使用：

```
$ md5sum -c *.md5
```

与md5sum类似，SHA-1是另一种常用的校验和算法。它从给定的输入文件中生成一个长度为40个字符的十六进制串。用来计算SAH-1串的命令是sha1sum。其用法和md5sum的非常相似。只需要把先前讲过的那些命令中的md5sum替换成sha1sum就行了，记住将输入文件名从file_sum.md5改为file_sum.sha1。

校验和对于核实下载文件的完整性非常有帮助。我们从网上下载的ISO镜像文件一般更容易出现错误（见图2-1）。为了检查接收文件正确与否，校验和得以广泛应用。校验和程序对同样的文件数据始终生成一模一样的校验和。

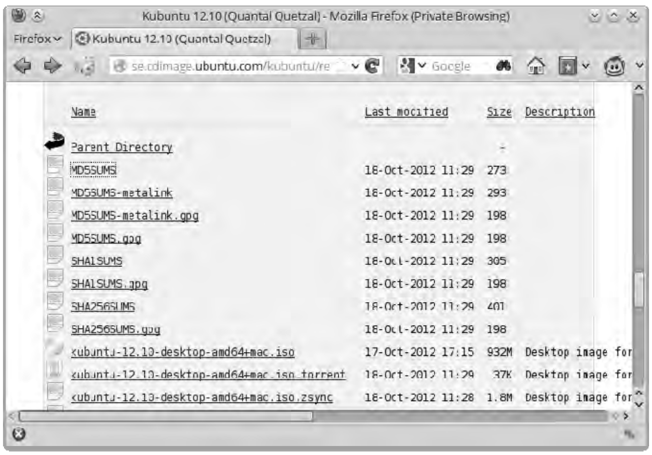


图　2-1

图2-2是md5sum生成的校验和。

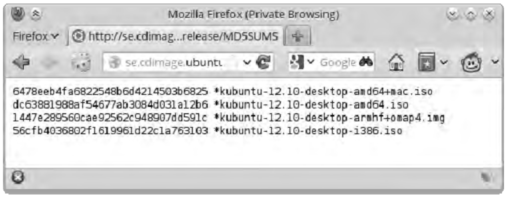


图　2-2

2.7.4　补充内容

对于多个文件，校验和同样可以发挥作用。现在就看看如何校验并核实多个文件。

对目录进行校验

校验和是从文件中计算得来的。对目录计算校验和意味着我们需要对目录中的所有文件以递归的方式进行计算。

这可以用命令`md5deep`或`sha1deep`来实现。首先，需要安装md5deep软件包以确保能找到这些命令。该命令的用法如下：

```
$ md5deep -rl directory_path > directory.md5
# -r使用递归的方式
# -l使用相对路径。默认情况下，md5deep会输出文件的绝对路径
```

或者也可以结合`find`来递归计算校验和：

```
$ find directory_path -type f -print0 | xargs -0 md5sum >> directory.md5
```

用下面的命令进行核实：

```
$ md5sum -c directory.md5
```

2.8　加密工具与散列

加密技术主要用于防止数据遭受未经授权的访问。加密算法有很多，我们会着重讲解那些常见的标准加密算法。在Linux环境下有一些工具可以用来执行加密和解密。有时我们使用加密算法散列值来验证数据的完整性。本节将介绍一些常用的加密工具以及这些工具所涉及的算法。

实战演练

让我们看看`crypt`、`gpg`、`base64`、`md5sum`、`sha1sum`以及`openssl`的用法。

- crypt是一个简单的加密工具，它从stdin接受一个文件以及口令作为输入，然后将加密数据输出到Stdout（因此要对输入、输出文件使用重定向）。

```
$ crypt <input_file >output_file
Enter passphrase:
```

它会要求输入一个口令。我们也可以通过命令行参数来提供口令。

```
$ crypt PASSPHRASE <input_file >encrypted_file
```

如果需要解密文件，可以使用：

```
$ crypt PASSPHRASE -d <encrypted_file >output_file
```

- gpg（GNU隐私保护）是一种应用广泛的工具，它使用加密技术来保护文件，以确保数据在送达目的地之前无法被读取。这里我们讨论如何加密、解密文件。

> gpg签名同样广泛用于在电子邮件通信中的邮件“签名”，以证明发送方的真实性。

用gpg加密文件：

```
$ gpg -c filename
```

该命令采用交互方式读取口令，并生成filename.gpg。使用以下命令解密gpg文件：

```
$ gpg filename.gpg
```

该命令读取口令，然后对文件进行解密。

> 本书并没有涉及gpg的过多细节。如果你感兴趣，希望进一步了解，请访问http://en.wikipedia.org/wiki/GNU_Privacy_Guard。

- Base64是一组相似的编码方案，它将ASCII字符转换成以64为基数的形式（radix-64 representation），以可读的ASCII字符串来描述二进制数据。base64命令可以用来编码/解码Base64字符串。要将文件编码为Base64格式，可以使用：

```
$ base64 filename > outputfile
```

或者

```
$ cat file | base64 > outputfile
```

base64可以从stdin中进行读取。

解码Base64数据：

```
$ base64 -d file > outputfile
```

或者

```
$ cat base64_file | base64 -d > outputfile
```

- md5sum与sha1sum都是单向散列算法，均无法逆推出原始数据。它们通常用于验证数据完整性或为特定数据生成唯一的密钥：

```
$ md5sum file
8503063d5488c3080d4800ff50850dc9  file

$ sha1sum file
1ba02b66e2e557fede8f61b7df282cd0a27b816b  file
```

这种类型的散列算法是存储密码的理想方案。密码使用其对应的散列值来存储。如果某个用户需要进行认证，读取该用户提供的密码并转换成散列值，然后将其与之前存储的散列值进行比对。如果相同，用户就通过认证，被允许访问；否则，就会被拒绝访问。将密码以明文形式存储是件非常冒险的事，会面临安全风险。

> 尽管使用广泛，md5sum和SHA-1已不再安全。因为计算能力的攀升使其变得容易被破解。推荐使用`bcrypt`或`sha512sum`这类工具进行加密。详情可参看http://codahale.com/how-to-saftyly-store-a-password/。

- shadow-like散列（salt散列）

让我们看看如何为密码生成shadow-like salt散列。在Linux中，用户密码是以散列值形式存储在文件 /etc/shadow中的。该文件中的典型内容类似于下面这样：

```
test:$6$fG4eWdUi$ohTKOlEUzNk77.4S8MrYe07NTRV4M3LrJnZP9p.qc1bR5c.EcOruzPXfEu1uloB
FUa18ENRH7F70zhodas3cR.:14790:0:99999:7:::
```

该行中的`$6$fG4eWdUi$ohTKOlEUzNk77.4S8MrYe07NTRV4M3LrJnZP9p.qc1bR5c.EcOruzPXfEu1uloBFUa18ENRH7F70zhodas3cR`是密码对应的shadow散列值。

某些情况下，我们得编写一些至关重要的管理脚本，这些脚本也许需要编辑密码，或是需要用shell脚本手动添加用户。这时我们必须生成shadow密码字符串，并向shadow文件中写入与上面类似的文本行。下面让我们看看如何用`openssl`生成shadow密码。

shadow密码通常都是salt密码。所谓`SALT`就是额外的一个字符串，用来起一个混淆的作用，使加密更加不易被破解。salt由一些随机位组成，被用作密钥生成函数的输入之一，以生成密码的salt散列值。

> 关于salt的更多细节信息，请参考维基百科页面http://en.wikipedia.org/wiki/Salt_(cryptography)。

```
$ opensslpasswd -1 -salt SALT_STRING PASSWORD
$1$SALT_STRING$323VkWkSLHuhbt1zkSsUG.
```

将SALT_STRING替换为随机字符串，并将PASSWORD替换成你想要使用的密码。

2.9 排序、唯一与重复

同文本文件打交道时，少不了要用到排序。sort命令能够帮助我们对文本文件和stdin进行排序操作。它通常会配合其他命令来生成所需要的输出。uniq是一个经常与sort一同使用的命令。它的作用是从文本或stdin中提取唯一（或重复）的行。这则攻略将演示sort和uniq命令的常见用法。

2.9.1 预备知识

sort命令既可以从特定的文件，也可以从stdin中获取输入，并将输出写入stdout。uniq的工作方式和sort一样。

2.9.2 实战演练

(1) 我们可以按照下面的方式轻松地对一组文件（例如file1.txt和file2.txt）进行排序：

```
$ sort file1.txt file2.txt > sorted.txt
```

或是

```
$ sort file1.txt file2.txt -o sorted.txt
```

(2) 按照数字顺序进行排序：

```
$ sort -n file.txt
```

(3) 按照逆序进行排序：

```
$ sort -r file.txt
```

(4) 按照月份进行排序（依照一月，二月，三月……）：

```
$ sort -M months.txt
```

(5) 合并两个已排序过的文件：

```
$ sort -m sorted1 sorted2
```

(6) 找出已排序文件中不重复的行：

```
$ sort file1.txt file2.txt | uniq
```

(7) 检查文件是否已经排序过：

```
#!/bin/bash
#功能描述：排序
sort -C filename ;
if [ $? -eq 0 ]; then
   echo Sorted;
else
   echo Unsorted;
fi
```

将filename替换成你需要进行检查的文件名，然后运行该脚本。

2.9.3　工作原理

sort命令包含大量的选项，能够对文件数据进行各种排序。如果使用uniq命令，那sort更是必不可少，因为前者要求输入数据必须经过排序。

sort和uniq可以应用于多种场景。让我们来看一下这些命令的各种选项及用法。

要检查文件是否排序过，可以采用以下方法：如果文件已经排序，sort会返回为0的退出码（$?），否则返回非0。

2.9.4　补充内容

我们已经介绍了sort命令的基本用法。下面来看看如何利用sort来完成一些复杂的任务。

1. 依据键或列进行排序

我们可以按列将下面的文本排序：

```
$ cat data.txt
1  mac    2000
2  winxp    4000
3  bsd    1000
4  linux    1000
```

有很多方法可以对这段文本排序。目前它是按照序号（第一列）来排序的。我们也可以依据第二列和第三列来排序。

-k指定了排序应该按照哪一个键（key）来进行。键指的是列号，而列号就是执行排序时的依据。-r告诉sort命令按照逆序进行排序。例如：

```
# 依据第1列，以逆序形式排序
$ sort -nrk 1  data.txt
4  linux    1000
3  bsd    1000
2  winxp    4000
1  mac    2000
# -nr表明按照数字，采用逆序形式排序

# 依据第2列进行排序
$ sort -k 2  data.txt
3  bsd    1000
4  linux    1000
1  mac    2000
2  winxp    4000
```

> 一定要留意用于按数字顺序进行排序的选项-n。sort命令对于字母表排序和数字排序有不同的处理方式。因此，如果要采用数字顺序排序，就应该明确地给出-n选项。

通常在默认情况下，键就是文本文件中的列。列与列之间用空格分隔。但有时候，我们需要将特定范围内的一组字符（例如，key1=character4-character8）作为键。在这种情况下，必须明确地将键指定为某个范围内的字符，这个范围可以用键起止的字符位置来表明。例如：

```
$ cat data.txt
1010hellothis
2189ababbba
7464dfddfdfd
$ sort -nk 2,3 data.txt
```

把醒目的字符作为数值键。为了提取这个键，用字符在行内的起止位置作为键的书写格式（在上面的例子中，起止位置是2和3）。

用第一个字符作为键：

```
$ sort -nk 1,1 data.txt
```

为了使sort的输出与以\0作为终止符的xargs命令相兼容，采用下面的命令：

```
$ sort -z data.txt | xargs -0
# 终止符\0用来保证xargs命令的使用安全
```

有时文本中可能会包含一些像空格之类的不必要的多余字符。如果需要忽略这些字符，并以字典序进行排序，可以使用：

```
$ sort -bd unsorted.txt
```

其中，选项-b用于忽略文件中的前导空白行，选项-d用于指明以字典序进行排序。

2. uniq

uniq命令通过消除重复内容，从给定输入中（stdin或命令行参数文件）找出唯一的行。它也可以用来找出输入中出现的重复行。

uniq只能作用于排过序的数据输入，因此，uniq要么使用管道，要么将排过序的文件作为输入，与sort命令结合使用。

你可以按照下面的方式从给定的输入数据中生成唯一的行（所谓“唯一的行”是指来自输入的所有行都会被打印出来，但是其中重复的行只会被打印一次）：

```
$ cat sorted.txt
bash
foss
hack
hack

$ uniq sorted.txt
bash
foss
hack
```

或是

```
$ sort unsorted.txt | uniq
```

只显示唯一的行（在输入文件中没有重复出现的行）：

```
$ uniq -u sorted.txt
bash
foss
```

或是

```
$ sort unsorted.txt | uniq -u
```

要统计各行在文件中出现的次数，使用下面的命令：

```
$ sort unsorted.txt | uniq -c
      1 bash
      1 foss
      2 hack
```

找出文件中重复的行：

```
$ sort unsorted.txt  | uniq -d
hack
```

我们可以结合-s和-w来指定键：

- ❑ -s 指定可以跳过前*n*个字符；
- ❑ -w 指定用于比较的最大字符数。

这个对比键被用作uniq操作的索引：

```
$ cat data.txt
u:01:gnu
d:04:linux
u:01:bash
u:01:hack
```

我们需要使用醒目的字符作为唯一的键。可以通过忽略前2个字符（-s 2），使用 -w选项（-w 2）指定用于比较的最大字符数。

```
$ sort data.txt | uniq -s 2 -w 2
d:04:linux
u:01:bash
```

我们将命令输出作为xargs命令的输入时，最好为输出的各行添加一个0值字节（zero-byte）终止符。在将uniq命令的输入作为xargs的数据源时，同样应当如此。如果没有使用0值字节终止符，那么在默认情况下，xargs命令会用空格作为定界符分割参数。例如，来自stdin的文本行“this is a line”会被xargs当做包含4个不同的参数，但实际上它只是一个单行而已。如果使用0值字节终止符，那么\0就被作为定界符，此时，包含空格的单行就能够被正确地解析成单个参数。

用uniq命令生成包含0值字节终止符的输出：

```
$ uniq -z file.txt
```

下面的命令将删除所有指定的文件，这些文件的名字是从files.txt中读取的：

```
$ uniq -z file.txt | xargs -0 rm
```

如果某个文件名在文件中出现多次，uniq命令只会将这个文件名写入stdout一次。

2.10　临时文件命名与随机数

编写shell脚本时，我们经常需要存储临时数据。最适合存储临时数据的位置是 /tmp（该目录中的内容在系统重启后会被清空）。有两种方法可以为临时数据生成标准的文件名。

2.10.1　实战演练

执行下列步骤来创建临时文件并进行不同的命名操作。

(1) 创建临时文件：

```
$ filename=`mktemp`
$ echo $filename
/tmp/tmp.8xvhkjF5fH
```

上面的代码创建了一个临时文件，并打印出存储在`$filename`中的文件名。

(2) 创建临时目录：

```
$ dirname=`mktemp -d`
$ echo $dirname
tmp.NI8xzW7VRX
```

上面的代码创建了一个临时目录，并打印出存储在`$dirname`中的目录名。

(3) 如果仅仅是想生成文件名，又不希望创建实际的文件或目录，方法如下：

```
$ tmpfile=`mktemp -u`
$ echo $tmpfile
/tmp/tmp.RsGmilRpcT
```

文件名被存储在`$tmpfile`中，但并没有创建对应的文件。

(4) 根据模板创建临时文件名：

```
$mktemp test.XXX
test.2tc
```

2.10.2 工作原理

`mktemp`命令的用法非常简单。它生成一个临时文件并返回其文件名（如果创建的是目录，则返回目录名）。

如果提供了定制模板，`X`会被随机的字符（字母或数字）替换。注意，`mktemp`正常工作的前提是保证模板中只少要有3个`X`。

2.11 分割文件和数据

在某些情况下，必须把文件分割成多个更小的片段。早期像软盘之类的设备容量都很有限，将文件分割得更小些，以便能够存储到多张磁盘中就显得至关重要了。不过如今我们分割文件就是出于其他目的了，比如为提高可读性、生成日志、通过E-mail发送文件等。在这则攻略中我们会看到将文件分割成不同大小的多种方法。

2.11.1　工作原理

假设有一个叫做data.file的测试文件，其大小为100KB。你可以将该文件分割成多个大小为10KB的文件，方法如下：

```
$ split -b 10k data.file
$ ls
data.file  xaa  xab  xac  xad  xae  xaf  xag  xah  xai  xaj
```

上面的命令将data.file分割成多个文件，每一个文件大小为10KB。这些文件以xab、xac、xad的形式命名。这表明它们都有一个字母后缀。如果想以数字为后缀，可以另外使用-d参数。此外，使用 -a length可以指定后缀长度：

```
$ split -b 10k data.file -d -a 4
```

除了k（KB）后缀，我们还可以使用M（MB）、G（GB）、c（byte）、w（word）等后缀。

```
$ ls
data.file x0009  x0019  x0029  x0039  x0049  x0059  x0069  x0079
```

2.11.2　补充内容

来看看split命令的其他选项。

为分割后的文件指定文件名前缀

之前那些分割后的文件都有一个文件名前缀x。我们也可以通过提供一个前缀名来使用自己的文件名前缀。split命令最后一个参数是PREFIX，其格式如下：

```
$ split [COMMAND_ARGS] PREFIX
```

我们加上文件名前缀再运行先前那个命令来分割文件：

```
$ split -b 10k data.file -d -a 4 split_file
$ ls
data.file       split_file0002  split_file0005  split_file0008  strtok.c
split_file0000  split_file0003  split_file0006  split_file0009
split_file0001  split_file0004  split_file0007
```

如果不想按照数据块大小，而是需要根据行数来分割文件的话，可以使用 -l no_of_lines：

```
$ split -l 10 data.file
# 分割成多个文件，每个文件包含10行
```

另一个有趣的的工具是csplit。它能够依据指定的条件和字符串匹配选项对日志文件进行分割。来看看这个工具是如何运作的。

csplit是split工具的一个变体。split只能够根据数据大小或行数分割文件，而csplit可以根据文本自身的特点进行分割。是否存在某个单词或文本内容都可作为分割文件的条件。

看一个日志文件示例：

```
$ cat server.log
SERVER-1
[connection] 192.168.0.1 success
[connection] 192.168.0.2 failed
[disconnect] 192.168.0.3 pending
[connection] 192.168.0.4 success
SERVER-2
[connection] 192.168.0.1 failed
[connection] 192.168.0.2 failed
[disconnect] 192.168.0.3 success
[connection] 192.168.0.4 failed
SERVER-3
[connection] 192.168.0.1 pending
[connection] 192.168.0.2 pending
[disconnect] 192.168.0.3 pending
[connection] 192.168.0.4 failed
```

我们需要将这个日志文件分割成server1.log、server2.log和server3.log，这些文件的内容分别取自原文件中不同的SERVER部分。那么，可以使用下面的方法来实现：

```
$ csplit server.log /SERVER/ -n 2 -s {*}  -f server -b "%02d.log"  ; rm server00.log

$ ls
server01.log  server02.log  server03.log  server.log
```

有关这个命令的详细说明如下。

- /SERVER/ 用来匹配某一行，分割过程即从此处开始。
- /[REGEX]/ 表示文本样式。包括从当前行（第一行）直到（但不包括）包含“SERVER”的匹配行。
- {*} 表示根据匹配重复执行分割，直到文件末尾为止。可以用{整数}的形式来指定分割执行的次数。
- -s 使命令进入静默模式，不打印其他信息。
- -n 指定分割后的文件名后缀的数字个数，例如01、02、03等。
- -f 指定分割后的文件名前缀（在上面的例子中，server就是前缀）。
- -b 指定后缀格式。例如%02d.log，类似于C语言中printf的参数格式。在这里文件名=前缀+后缀=server + %02d.log。

因为分割后的第一个文件没有任何内容（匹配的单词就位于文件的第一行中），所以我们删除了server00.log。

2.12　根据扩展名切分文件名

有一些脚本是依据文件名进行各种处理的。我们可能会需要在保留扩展名的同时修改文件名、转换文件格式（保留文件名的同时修改扩展名）或提取部分文件名。shell所具有的一些内建功能可以依据不同的情况来切分文件名。

2.12.1　实战演练

借助%操作符可以轻松将名称部分从“*名称.扩展名*”这种格式中提取出来。你可以按照下面的方法从sample.jpg中提取名称。

```
file_jpg="sample.jpg"
name=${file_jpg%.*}
echo File name is: $name
```

输出结果：

```
File name is: sample
```

下一个任务是将文件名的扩展名部分提取出来，这可以借助 # 操作符实现。

提取文件名中的 .jpg并存储到变量`file_jpg`中：

```
extension=${file_jpg#*.}
echo Extension is: jpg
```

输出结果：

```
Extension is: jpg
```

2.12.2　工作原理

在第一个任务中，为了从“*名称.扩展名*”这种格式中提取名称，我们使用了%操作符。

`${VAR%.*}` 的含义如下所述：

- 从 `$VAR`中删除位于 % 右侧的通配符（在前例中是`.*`）所匹配的字符串。通配符从右向左进行匹配。
- 给`VAR`赋值，`VAR=sample.jpg`。那么，通配符从右向左就会匹配到.jpg，因此，从 `$VAR`中删除匹配结果，就会得到输出`sample`。

%属于非贪婪（non-greedy）操作。它从右到左找出匹配通配符的最短结果。还有另一个操作符 %%，这个操作符与%相似，但行为模式却是贪婪的，这意味着它会匹配符合条件的最长的字符

串。例如，我们现在有这样一个文件：

```
VAR=hack.fun.book.txt
```

使用%操作符：

```
$ echo ${VAR%.*}
```

得到输出：hack.fun.book

操作符%使用.*从右向左执行非贪婪匹配（.txt）。

使用%%操作符：

```
$ echo ${VAR%%.*}
```

得到输出：hack

操作符%%则用.*从右向左执行贪婪匹配（.fun.book.txt）。

在第二个任务中，我们用 # 操作符从文件名中提取扩展名。这个操作符与%类似，不过求值方向是从左向右。

${VAR#*.}的含义如下所述：

> 从$VAR中删除位于#右侧的通配符（即在前例中使用的*.）所匹配的字符串。通配符从左向右进行匹配。

和 %% 类似，#也有一个相对应的贪婪操作符 ##。

从左向右进行贪婪匹配，并从指定变量中删除匹配结果。

来看一个例子：

```
VAR=hack.fun.book.txt
```

使用#操作符：

```
$ echo ${VAR#*.}
```

得到输出：fun.book.txt

操作符#用*.从左向右执行非贪婪匹配（hack）。

使用##操作符：

```
$ echo ${VAR##*.}
```

得到输出：txt

操作符##则用*.从左向右执行贪婪匹配（txt）。

[因为文件名中可能包含多个'.'字符，所以相较于#，## 更适合于从文件名中提取扩展名。## 执行的是贪婪匹配，因而总是能够准确地提取出扩展名。]

这里有个能够提取域名不同部分的实用案例。假定URL="www.google.com"：

```
$ echo ${URL%.*} # 移除.*所匹配的最右边的内容
www.google

$ echo ${URL%%.*} # 将从右边开始一直匹配到最左边的*.移除（贪婪操作符）
www


$ echo ${URL#*.} # 移除*.所匹配的最左边的内容
google.com


$ echo ${URL##*.} # 将从左边开始一直匹配到最右边的*.移除（贪婪操作符）
com
```

2.13　批量重命名和移动

重命名多个文件是我们经常会碰到的一项工作。举个简单的例子，当你把照片从数码相机传输到你的计算机之后，你可能会删除其中某些不如意的部分，这会使图像文件的编号变得不再连续。你会想使用特定的前缀和连续的数字对它们进行重命名。我们当然可以借助第三方软件执行这类重命名操作，但是也可以使用Bash命令在短短几秒钟之内完成同样的活儿。

另一件经常要做的工作是，将文件名中包含某个特定部分（例如相同的前缀）或者具有特定类型的所有文件移动到指定的文件夹中。让我们看看如何用脚本来执行这些操作。

2.13.1　预备知识

rename命令利用Perl正则表达式修改文件名。综合运用find、rename和mv，我们能做到的事其实很多。

2.13.2　实战演练

用特定的格式重命名当前目录下的图像文件，最简单的方法是使用下面的脚本：

```
#!/bin/bash
#文件名: rename.sh
#用途: 重命名 .jpg 和 .png 文件

count=1;
for img in `find . -iname '*.png' -o -iname '*.jpg' -type f -maxdepth 1`
do
  new=image-$count.${img##*.}

  echo "Renaming $img to $new"
  mv "$img" "$new"
  let count++

done
```

输出如下：

```
$ ./rename.sh
Renaming hack.jpg to image-1.jpg
Renaming new.jpg to image-2.jpg
Renaming next.png to image-3.png
```

该脚本将当前目录下所有的.jpg和.png文件重命名，新文件名的格式为image-1.jpg、image-2.jpg、image-3.jpg、image-4.png等，依次类推。

2.13.3 工作原理

在前面的重命名脚本中，使用了`for`循环对所有扩展名为.jpg或.png的文件进行迭代。我们使用`find`命令进行搜索，选项`-o`用于指定多个`-iname`选项，后者用于执行大小写无关的匹配。借助`-maxdepth 1`，确保`$img`中只包含来自当前目录的文件名，无视其他的子目录。

为了跟踪图像编号，我们初始化变量`count=1`。下一步就是用mv命令重命名文件。因此需要构造出新的文件名。`${img##*.}`对处于当前循环中的文件名进行解析并获得文件扩展名（请参看2.12节中对于`${img##*.}`的解释）。

`let count++`用来在每次循环中增加文件编号。

还有许多其他执行重命名操作的方法，来看其中一些例子。

- 将`*.JPG`更名为`*.jpg`：

  ```
  $ rename *.JPG *.jpg
  ```

- 将文件名中的空格替换成字符“_”：

  ```
  $ rename 's/ /_/g' *
  ```

's/ /_/g'用于替换文件名，而 * 是用于匹配目标文件的通配符，它也可以以 *.txt或其他样式出现。

- 转换文件名的大小写：

```
$ rename 'y/A-Z/a-z/' *
$ rename 'y/a-z/A-Z/' *
```

- 将所有的 .mp3文件移入给定的目录：

```
$ find path -type f -name "*.mp3" -exec mv {} target_dir \;
```

- 将所有文件名中的空格替换为字符“_”：

```
$ find path -type f -exec rename 's/ /_/g' {} \;
```

2.14　拼写检查与词典操作

Linux大多数发行版都含有一份词典文件。然而，我发现几乎没人在意过这个文件，更别提利用它了。还有一个叫做aspell的命令行工具，其作用是进行拼写检查。让我们通过一些脚本来看看如何使用词典文件和拼写检查工具。

2.14.1　实战演练

目录 /usr/share/dict/ 包含了一些词典文件。“词典文件”就是包含了词典单词列表的文本文件。我们可以利用这个列表来检查某个单词是否为词典中的单词。

```
$ ls /usr/share/dict/
american-english  british-english
```

为了检查给定的单词是否为词典中的单词，可以使用下面的脚本：

```
#!/bin/bash
#文件名：checkword.sh
word=$1
grep "^$1$" /usr/share/dict/british-english -q
if [ $? -eq 0 ]; then
  echo $word is a dictionary word;
else
  echo $word is not a dictionary word;
fi
```

这个脚本的用法如下：

```
$ ./checkword.sh ful
ful is not a dictionary word
```

```
$ ./checkword.sh fool
fool is a dictionary word
```

2.14.2 工作原理

在grep中，^ 标记着单词的开始，$ 标记着单词的结束。

-q 禁止产生任何输出。

另外，我们也可以用拼写检查命令aspell来核查某个单词是否在词典中：

```
#!/bin/bash
#文件名：aspellcheck.sh
word=$1

output=`echo \"$word\" | aspell list`

if [ -z $output ]; then
        echo $word is a dictionary word;
else
        echo $word is not a dictionary word;
fi
```

当给定的输入不是一个词典单词时，aspell list命令产生输出文本，反之则不产生任何输出。-z用于确认 $output是否为空。

列出文件中以特定单词起头的所有单词：

```
$ look word filepath
```

或者使用

```
$ grep "^word" filepath
```

在默认情况下，如果没有给出文件参数，look命令会使用默认词典（/usr/share/dict/words）并返回输出。

```
$look word
# 像这样使用时，look命令以默认词典作为文件参数
```

例如：

```
$ look android
android
android's
androids
```

2.15 交互输入自动化

就编写自动化工具或测试工具而言，实现命令的交互输入自动化极其重要。在很多情况下，我们要同一些以交互方式读取输入的命令打交道。下面的例子就是一个要求提供交互式输入的命令执行过程。

```
$ command
Enter a number: 1
Enter name : hello
You have entered 1,hello
```

2.15.1 预备知识

能够自动接受输入的自动化工具，对于本地命令或远程应用来说都有益处。让我们看看如何实现自动化。

2.15.2 实战演练

思考一下交互式输入的过程。参照之前的代码，我们可以将涉及的步骤描述如下：

```
1[Return]hello[Return]
```

观察实际通过键盘输入的字符，可以将上面的1、Return、hello以及Return转换为以下字符串：

```
"1\nhello\n"
```

按下回车键时会发送 \n。通过添加 \n，就可以得到发送给stdin的字符串。

因此，通过发送与用户输入等同的字符串，我们就可以实现在交互过程中自动发送输入。

2.15.3 工作原理

先写一个读取交互式输入的脚本，然后用这个脚本进行自动化的演示：

```
#!/bin/bash
#文件名: interactive.sh
read -p "Enter number:" no ;
read -p "Enter name:" name
echo You have entered $no, $name;
```

按照下面的方法向脚本自动发送输入：

```
$ echo -e "1\nhello\n" | ./interactive.sh
You have entered 1, hello
```

看来制作的输入生效了。

我们用echo -e来生成输入序列，-e表明echo会解释转义序列。如果输入内容比较多，那么可以用单独的输入文件结合重定向操作符来提供输入：

```
$ echo -e "1\nhello\n"  > input.data
$ cat input.data
1
hello
```

制作输入文件时，你也可以不用echo命令：

```
$ ./interactive.sh < input.data
```

这个方法是从文件中导入交互式输入数据。

如果你是逆向工程师，那可能同缓冲区溢出攻击打过交道。要实施攻击，我们需要将十六进制形式的shellcode（例如“\xeb\x1a\x5e\x31\xc0\x88\x46”）进行重定向。这些字符没法直接通过键盘输入，因为键盘上并没有其对应的按键。因此，我们应该使用：

```
echo -e \xeb\x1a\x5e\x31\xc0\x88\x46"
```

这条命令会将shellcode重定向到有缺陷的可执行文件中。

我们已经描述了一种方法，它通过stdin将所需的文本进行重定向，从而实现交互式输入程序自动化。但是我们并没有检查所发送的输入内容。我们期望程序以特定（固定）的次序处理我们所发送的输入。如果程序对于输入采取随机或其他处理次序，或者甚至不要求输入某些内容，那么之前的方法就要出问题了。它会发送不符合程序要求的错误输入。为了处理动态输入并通过检查程序运行时的输入需求来提供输入内容，我们要使用一个出色的工具expect。expect命令可以根据输入要求提供适合的输入。

2.15.4　补充内容

我们看看expect的用法。交互式输入自动化也可以用其他方法实现。expect脚本就是其中之一。

用expect实现自动化

在默认情况下，多数常见的Linux发行版中并不包含expect。你得用软件包管理器手动进行安装。

expect等待特定的输入提示，通过检查输入提示来发送数据。

```
#!/usr/bin/expect
#文件名: automate_expect.sh
spawn ./interactive.sh
expect "Enter number:"
send "1\n"
expect "Enter name:"
send "hello\n"
expect eof
```

运行结果如下：

```
$ ./automate_expect.sh
```

在这个脚本中：

- spawn参数指定需要自动化哪一个命令；
- expect参数提供需要等待的消息；
- send是要发送的消息；
- expect eof指明命令交互结束。

2.16　利用并行进程加速命令执行

在过去的几年中，计算能力有了大幅度的攀升。但这并不是仅仅因为处理器有了更高的时钟频率，而是因为多核的出现。对于用户而言，这意味着单个物理处理器中包含了多个逻辑处理器。

除非软件能够善加利用多核，否则它们毫无用武之地。如果你有一个需要进行大量运算的程序，仅运行在其中一个核心上，那么其他的核心都会被闲置。如果想提高速度，软件必须留意并充分利用多核。

在这则攻略中，我们会看到如何让命令运行得更快。

2.16.1　实战演练

以之前讲过的md5sum命令为例。由于涉及运算，该命令属于CPU密集型命令。如果多个文件需要生成校验和，我们可以使用下面的脚本来运行md5sum的多个实例。

```
#/bin/bash
#文件名: generate_checksums.sh
PIDARRAY=()
for file in File1.iso File2.iso
do
  md5sum $file &
```

```
  PIDARRAY+=("$!")
done
wait ${PIDARRAY[@]}
```

运行脚本后，可以得到如下输出：

```
$ ./generate_checksums.sh
330dcb53f253acdf76431cecca0fefe7  File1.iso
bd1694a6fe6df12c3b8141dcffaf06e6  File2.iso
```


输出结果和下面命令的结果一样：

```
md5sum File1.iso File2.iso
```

但是因为多个md5sum命令是同时运行的，如果你使用的是多核处理器，就会更快地获得运行结果（可以使用time命令来验证）。

2.16.2 工作原理

我们利用了Bash的操作符&，它使得shell将命令置于后台并继续执行脚本。这意味着一旦循环结束，脚本就会退出，而md5sum命令仍在后台运行。为了避免这种情况，我们使用$!来获得进程的PID，在Bash中，$!保存着最近一个后台进程的PID。我们将这些PID放入数组，然后使用wait命令等待这些进程结束。

第3章 以文件之名

本章内容

- 生成任意大小的文件
- 文本文件的交集与差集
- 查找并删除重复文件
- 文件权限、所有权和粘滞位
- 创建不可修改文件
- 批量生成空白文件
- 查找符号链接及其指向目标
- 列举文件类型统计信息
- 使用环回文件
- 生成ISO文件及混合ISO
- 查找文件差异并进行修补
- `head`与`tail`——打印文件的前10行或后10行
- 只列出目录的其他方法
- 在命令行中用`pushd`和`popd`快速定位
- 统计文件的行数、单词数和字符数
- 打印目录树

3.1 简介

Unix将操作系统中的一切都视为文件。所有操作都离不开文件，可以利用它们进行各种与系统或进程相关的处理工作。例如，我们所使用的命令终端就是和一个设备文件关联在一起的。我们可以通过写入特定终端所对应的设备文件来实现向终端写入信息。文件的形式多种多样，比如目录、普通文件、块设备、字符设备、符号链接、套接字和命名管道等。文件名、大小、文件类型、文件内容修改时间、文件访问时间、文件属性更改时间、i节点、链接以及文件所在的文件系统等都是文件的属性。本章包含的实战攻略涉及文件相关的操作及属性。

3.2 生成任意大小的文件

出于各种原因，你也许需要生成一个包含随机数据的文件。这可能是用于执行测试的测试文件，比如用一个大文件作为输入来测试应用程序的效率；也可能是测试文件分割，或是创建环回

文件系统（**环回文件**自身包含文件系统，这种文件可以像物理设备一样使用mount命令进行挂载）。专门写一个程序来创建这些文件可得花点功夫，所以我们用一些通用工具帮忙解决。

实战演练

创建特定大小的大文件最简单的方法就是利用dd命令。dd命令会克隆给定的输入内容，然后将一模一样的一份副本写入到输出。stdin、设备文件、普通文件等都可作为输入，stdout、设备文件、普通文件等也可作为输出。下面是使用dd命令的一个示例：

```
$ dd if=/dev/zero of=junk.data bs=1M count=1
1+0 records in
1+0 records out
1048576 bytes (1.0 MB) copied, 0.00767266 s, 137 MB/s
```


该命令会创建一个1MB大小的文件junk.data。来看一下命令参数：if代表输入文件（input file），of代表输出文件（output file），bs代表以字节为单位的块大小（block size），count代表需要被复制的块数。

> 使用dd命令时一定得留意，该命令运行在设备底层。要是你不小心出了岔子，搞不好会把磁盘清空或是损坏数据。所以一定要反复检查dd命令所用的语法是否正确，尤其是参数of=。

在上面的例子中，我们将bs指定为1MB，count指定为1，于是得到了一个大小为1MB的文件。如果把bs设为2MB，count设为2，那么总文件大小就是4MB。

块大小可以使用各种计量单位。表3-1中任意一个字符都可以置于数字之后来指定字节数。

表　3-1

单元大小	代　码
字节（1B）	c
字（2B）	w
块（512B）	b
千字节（1024B）	k
兆字节（1024KB）	M
吉字节（1024MB）	G

按照这种方法，我们就可以生成任意大小的文件。表3-1中给出的计量单位都可以使用。

/dev/zero是一个字符设备，它会不断返回0值字节（\0）。

如果不指定输入参数（if），默认情况下dd会从stdin中读取输入。与之类似，如果不指定输出参数（of），则dd会将stdout作为默认输出。

也可以用dd命令传输大量数据并观察命令输出来测量内存的操作速度（例如，在前一个例子中所示的1048576 bytes (1.0 MB) copied, 0.00767266 s, 137 MB/s）。

3.3 文本文件的交集与差集

交集（intersection）和差集（set difference）操作在数学课上的集合论中经常会被用到。不过，有时候对字符串进行类似的操作也很有用。

3.3.1 预备知识

comm命令可用于两个文件之间的比较。它有很多不错的选项可用来调整输出，以便我们执行交集、求差（difference）以及差集操作[①]。

- 交集：打印出两个文件所共有的行。
- 求差：打印出指定文件所包含的且互不相同的那些行。
- 差集：打印出包含在文件A中，但不包含在其他指定文件中的那些行。

3.3.2 实战演练

需要注意的是comm必须使用排过序的文件作为输入。请看看下面的例子：

```
$ cat A.txt
apple
orange
gold
silver
steel
iron

$ cat B.txt
orange
gold
cookies
carrot
```

① 假设现在有两个文件A和B，内容分别是：A(1,2,3)，B(3,4,5)。那么，对这两个文件进行操作的结果如下。
交集：3。
求差：1,2,4,5。
差集（A）：1,2。

```
$ sort A.txt -o A.txt ; sort B.txt -o B.txt
```

(1) 首先执行不带任何选项的`comm`：

```
$ comm A.txt B.txt
apple
        carrot
        cookies
                gold
iron
                orange
silver
steel
```

输出的第一列包含只在A.txt中出现的行，第二列包含只在B.txt中出现的行，第三列包含A.txt和B.txt中相同的行。各列以制表符（`\t`）作为定界符。

(2) 为了打印两个文件的交集，我们需要删除第一列和第二列，只打印出第三列：

```
$ comm A.txt B.txt -1 -2
gold
orange
```

(3) 打印出两个文件中不相同的行：

```
$ comm A.txt B.txt  -3
apple
        carrot
        cookies
iron
silver
steel
```

在这次的输出中，那些唯一出现的行使得列中出现了空白字段。所以这两列在同一行上不会同时都出现内容。为了提高输出结果的可用性，需要删除空白字段，将两列合并成一列：

```
apple
carrot
cookies
iron
silver
steel
```

(4) 要生成规范的输出，得使用下面的命令：

```
$ comm A.txt B.txt  -3 | sed 's/^\t//'
apple
```

```
carrot
cookies
iron
silver
steel
```

(5) 通过删除不需要的列，我们就可以分别得到A.txt和B.txt的差集。

- A.txt的差集

```
$ comm A.txt B.txt -2 -3
```

`-2 -3` 删除第二列和第三列。

- B.txt的差集

```
$ comm A.txt B.txt -1 -3
```

`-1 -3` 删除第一列和第三列。

3.3.3　工作原理

comm的命令行选项可以按照需求对输出进行格式化，例如：

- `-1` 从输出中删除第一列；
- `-2` 从输出中删除第二列；
- `-3` 从输出中删除第三列。

在生成统一输出时，sed命令通过管道获取comm的输出。它删除行首的 `\t`字符。sed中的s表示替换（substitute）。`/^\t/` 匹配行前的 `\t`（`^`是行首标记）。`//`（两个`/`操作符之间没有任何字符）是用来替换行首的`\t`的字符串。如此一来，就删除了所有行首的`\t`。

差集操作允许你比较两个文件，打印出只在A.txt或B.txt中出现的行。当A.txt和B.txt作为comm命令的参数时，输出中的第一列是A.txt相对于B.txt的差集，第二列是B.txt相对于A.txt的差集。

3.4　查找并删除重复文件

重复文件是同一个文件的多个副本。有时候我们需要删除重复的文件，只保留其中一份。通过查看文件内容来识别重复文件是件挺有意思的活儿。可以结合多种shell工具来完成这项任务。在这则攻略中，我们讨论如何查找重复文件并根据查找结果执行相关的操作。

3.4.1　预备知识

我们可以通过比较文件内容来识别它们。校验和是依据文件内容来计算的，内容相同的文件

自然会生成相同的校验和，因此，我们可以通过比较校验和来删除重复文件。

3.4.2 实战演练

(1) 按照下面的方法创建一些测试文件：

```
$ echo "hello" > test ; cp test test_copy1 ; cp test test_copy2;
$ echo "next" > other;
# test_copy1和test_copy2都是test文件的副本
```

(2) 删除重复文件的脚本代码如下：

```
# !/bin/bash
# 文件名：remove_duplicates.sh
# 用途：查找并删除重复文件，每一个文件只保留一份

ls -lS --time-style=long-iso | awk 'BEGIN {
  getline; getline;
  name1=$8; size=$5
}
{
  name2=$8;
  if (size==$5)
  {
    "md5sum "name1 | getline; csum1=$1;
    "md5sum "name2 | getline; csum2=$1;
    if ( csum1==csum2 )
    {
      print name1; print name2
    }
  };

  size=$5; name1=name2;
}' | sort -u > duplicate_files


cat duplicate_files | xargs -I {} md5sum {} | sort | uniq -w 32 | awk '{ print
"^"$2"$" }' | sort -u >  duplicate_sample

echo Removing..
comm duplicate_files duplicate_sample  -2 -3 | tee /dev/stderr | xargs rm
echo Removed duplicates files successfully.
```

(3) 执行方式：

```
$ ./remove_duplicates.sh
```

3.4.3 工作原理

前文中的shell脚本会找出某个目录中同一文件的所有副本，然后保留单个副本的同时删除其他副本。让我们研究一下这个脚本的工作原理。

`ls -lS`对当前目录下的所有文件按照文件大小进行排序，并列出文件的详细信息。`awk`读取`ls -lS`的输出，对行列进行比较，找出重复文件。

下面是代码的执行逻辑。

- 我们将文件依据大小排序并列出，这样大小接近的文件就会排列在一起。识别大小相同的文件是我们查找重复文件的第一步。接下来，计算这些文件的校验和。如果校验和相同，那么这些文件就是重复文件，将被删除。
- 在从文件中读取文本行之前，首先要执行`awk`的`BEGIN{}`语句块。读取文本行的工作在`{}`语句块中进行，读取并处理完所有的行之后，执行`END{}`语句块。`ls -lS`的输出如下：

```
total 16
-rw-r--r-- 1 slynux slynux 5 2010-06-29 11:50 other
-rw-r--r-- 1 slynux slynux 6 2010-06-29 11:50 test
-rw-r--r-- 1 slynux slynux 6 2010-06-29 11:50 test_copy1
-rw-r--r-- 1 slynux slynux 6 2010-06-29 11:50 test_copy2
```

- 第1行输出告诉我们文件数量，这个信息在本例中没什么用处。我们用`getline`读取第1行，然后丢弃。由于需要对每一行及其下一行来比对文件大小，因此用`getline`读取长文件列表的第一行，并存储文件名和大小（它们分别是第8列和第5列）。这样我们就先得到了一行。接下来，`awk`进入`{}`语句块（在这个语句块中读取余下的文本行），读取到的每一行文本都会执行该语句块。它将当前行中读取到的文件大小与之前存储在变量`size`中的值进行比较。如果相等，那就意味着两个文件至少在大小上是相同的，随后再用`md5sum`执行进一步的检查。

我们在给出的解决方法中使用了一些技巧。

在`awk`中，外部命令的输出可以用下面的方法读取：

```
"cmd"| getline
```

随后就可以在`$0`中获取命令的输出，在`$1`,`$2`,…`$n`中获取命令输出中的每一列。我们将文件的md5sum保存在变量`csum1`和`csum2`中。变量`name1`和`name2`保存文件列表中位置连续的文件名。如果两个文件的校验和相同，那它们肯定是重复文件，其文件名会被打印出来。

我们需要从每组重复文件中找出一个文件，这样就可以删除其他副本了。计算重复文件的`md5sum`，从每一组重复文件中打印出其中一个。这是通过`-w 32`比较每一行的`md5sum`（`md5sum`

输出中的前32个字符，md5sum的输出通常由32个字符的散列值和文件名组成），然后找出那些不相同的行。这样，每组重复文件中的一个采样就被写入duplicate_sample。

现在需要将duplicate_files中列出的、且未包含在duplicate_sample之内的全部文件删除。这些文件由comm命令负责打印出来。我们可以使用差集操作来实现（参考3.3节）。

comm通常只接受排序过的文件。所以，在重定向到duplicate_files和duplicate_sample之前，首先用sort -u作为一个过滤器。

tee命令在这里有一个妙用：它在将文件名传递给rm命令的同时，也起到了print的作用。tee将来自stdin的行写入文件，同时将其发送到stdout。我们也可以将文本重定向到stderr来实现终端打印功能。/dev/stderr是对应于stderr（标准错误）的设备。通过重定向到stderr设备文件，来自stdin的文本将会以标准错误的形式出现在终端中。

3

3.5 文件权限、所有权和粘滞位

文件权限和所有权是Unix/Linux文件系统（如ext文件系统）最显著的特性之一。在Unix/Linux平台工作时，经常会碰到与文件权限及所有权相关的问题。这则攻略就考查了文件权限和所有权的各种用例。

Linux系统中的每一个文件都与多种类型的权限相关联。在这些权限中，我们通常要和三类权限打交道（用户、用户组以及其他用户）。

用户（user）是文件的所有者。**用户组**（group）是多个用户的集合（由系统管理员指定），系统允许这些用户对文件进行某种形式的访问。**其他用户**（others）是除文件用户或用户组之外的任何人。

用命令ls -l可以列出文件的权限：

```
-rw-r--r-- 1 slynux slynux  2497  2010-02-28 11:22 bot.py
drwxr-xr-x 2 slynux slynux  4096  2010-05-27 14:31 a.py
-rw-r--r-- 1 slynux slynux  539   2010-02-10 09:11 cl.pl
```

第1列输出明确了后面的输出。其中第一个字母的对应关系如下所示。

- -—— 普通文件。
- d—— 目录。
- c—— 字符设备。
- b—— 块设备。
- l—— 符号链接。
- s—— 套接字。

- `p`—— 管道。

剩下的部分可以划分成三组，每组3个字符（--- --- ---）。第一组的3个字符（---）对应用户权限（所有者），第二组对应用户组权限，第三组对应其他用户权限。这9个字符（即9个权限）中的每一个字符指明是否其设置了某种权限。如果已设置，对应位置上会出现一个字符，否则出现一个`'-'`，表明没有设置对应的权限。

让我们来看一下每个字符组对于用户、用户组以及其他用户的含义。

- **用户**（权限序列：`rwx------`）：第一个字符指定用户是否拥有文件的读权限。如果为用户设置了读权限，`r`将出现在第一个字符的位置上。第二个字符指定了写（修改）权限（`w`），第三个字符指定了用户是否拥有执行权限（`x`，即运行该文件的权限）。可执行文件通常会设置执行权限。用户还有一个称为`setuid`（`S`）的特殊权限，它出现在执行权限（`x`）的位置。`setuid`权限允许用户以其拥有者的权限来执行可执行文件，即使这个可执行文件是由其他用户运行的。

  具有`setuid`权限的文件的权限序列如下：`-rwS------`。

  目录同样也有读、写、执行权限。不过对于目录来说，读、写、执行权限的含义有点不一样：

  - 目录的读权限（`r`）允许读取目录中文件和子目录的列表；
  - 目录的写权限（`w`）允许在目录中创建或删除文件或目录；
  - 目录的执行权限（`x`）指明是否可以访问目录中的文件和子目录。

- **用户组**（权限序列：`---rwx---`）：第二组字符指定了组权限。组权限的`rwx`的含义和用户权限中的一样。组权限并没有`setuid`，但是有一个`setgid`（`S`）位。它允许以同该目录拥有者所在组相同的有效组权限来允许可执行文件。但是这个组和实际发起命令的用户组未必相同。
  组权限的权限序列如下：`----rwS---`。
- **其他用户**（权限序列：`------rwx`）：最后三个字符是其他用户权限。和用户以及用户组一样，其他用户也有读、写、执行权限，但是并没有`S`权限（如`setuid`和`setgid`）。

目录有一个特殊的权限，叫做**粘滞位**（sticky bit）。如果目录设置了粘滞位，只有创建该目录的用户才能删除目录中的文件，即使用户组和其他用户也有写权限，也无能无力。粘滞位出现在其他用户权限中的执行权限（`x`）位置。它使用`t`或`T`来表示。如果没有设置执行权限，但设置了粘滞位，就使用`t`；如果同时设置了执行权限和粘滞位，就使用`T`。

例如：

```
------rwt , ------rwT
```

设置目录粘滞位的一个典型例子就是 /tmp。

在ls -l的每一行输出中，字符串slynux slynux分别对应所属用户和所属用户组。第一个slynux代表所属用户，第二个slynux代表用户组的所有者。

3.5.1 实战演练

可使用chmod命令设置文件权限。

假设需要设置权限：rwx rw- r--。

可以像下面这样使用chmod：

```
$ chmod u=rwx g=rw o=r filename
```

在这里：

- u——指定用户权限
- g——指定用户组权限
- o——指定其他实体权限

可以对用户、用户组和其他用户用 + 进行添加权限，用 - 删除权限。

文件已经具有权限rwx rw- r--，现在需要增加可执行权限，方法如下：

```
$ chmod o+x filename
```

给所有权限类别（即用户、用户组和其他用户）增加可执行权限：

```
$ chmod a+x filename
```

其中，a表示全部（all）。

如果需要删除权限，则使用 -，例如：

```
$ chmod a-x filename
```

也可以用八进制数来设置权限。权限由3位八进制数来表示，每一位按顺序分别对应用户、用户组和其他用户。

读、写和执行权限都有与之对应的唯一的八进制数：

- r-- = 4
- -w- = 2
- --x = 1

我们可以将权限序列的八进制值相加来获得所需的权限组合，例如：

- rw- = 4 + 2 = 6
- r-x = 4 + 1 = 5

权限序列 rwx rw- r--的数字表示形式如下：

- rwx = 4 + 2 + 1 = 7
- rw- = 4 + 2 = 6
- r-- = 4

因此，rwx rw- r-- 等于764，那么使用八进制值设置权限的命令为：

```
$ chmod 764 filename
```

3.5.2　补充内容

让我们再看一些其他有关文件和目录的操作。

1. 更改所有权

要更改文件的所有权，可以使用chown命令：

```
$ chown user.group filename
```

例如：

```
$ chown slynux.slynux test.sh
```

在这里，slynux既是用户名，也是用户组名。

2. 设置粘滞位

粘滞位是一种应用于目录的权限类型。通过设置粘滞位，使得只有目录的所有者才能够删除目录中的文件，即使用户组和其他用户拥有足够的权限也不能执行删除操作。

要设置粘滞位，利用chmod将 +t应用于目录：

```
$ chmod a+t directory_name
```

3. 以递归的方式设置权限

有时候需要以递归的方式修改当前目录下的所有文件和子目录的权限，方法如下：

```
$ chmod 777 . -R
```

选项 -R指定以递归的方式修改权限。

我们用“.”指定当前工作目录，这等同于：

```
$ chmod 777 "$(pwd)" -R.
```

4. 以递归的方式设置所有权

用chown命令结合 -R就可以以递归的方式设置所有权：

```
$ chown user.group . -R
```

5. 以不同的身份运行可执行文件

一些可执行文件需要以不同的用户身份（启动该文件的当前用户之外的用户），通过文件路径来执行（如 ./executable_name）。有一个叫做setuid的特殊文件权限，它允许其他用户以文件所有者的身份来执行文件。

首先将文件的所有权替换为需要执行该文件的用户，然后以该用户的身份登录。运行下面的命令：

```
$ chmod +s executable_file

# chown root.root executable_file
# chmod +s executable_file
$ ./executable_file
```

现在，这个文件实际上每次都是以root用户的身份来执行。

setuid的使用不是无限制的。为了确保安全，它只能应用在Linux ELF格式二进制文件上，而不能用于脚本文件。

3.6 创建不可修改的文件

在常见的Linux扩展文件系统中（如ext2、ext3、ext4等），可以借助某种文件属性将文件设置为不可修改（immutable）。一旦设置，任何用户（包括超级用户）都不能删除该文件，除非其不可修改的属性被移除。通过查看 /etc/mtab文件，我们很容易获知所有挂载分区的文件系统类型。这个文件的第一列指定了分区设备路径（如 /dev/sda5），第三列指定了文件系统类型（如ext3）。

不可修改属性是避免文件被篡改的安全手段之一。/etc/resolv.conf文件就是这样的一个例子。该文件包含了一组DNS服务器列表。DNS服务器负责将域名（例如packtpub.com）转换成IP地址。它通常被设置成你所属ISP的DNS服务器地址。但有些用户更喜欢使用第三方的DNS服务器，他们会修改/etc/resolv.conf，将其指向所选的服务器。可当下次你再连接到ISP时，/etc/resolv.conf又会恢复到之前的设置。为了避免这种情况，需要将/etc/resolv.conf 设置成不可修改。

3.6.1　预备知识

chattr能够将文件设置为不可修改。不过chattr能做的可不止这些。

3.6.2　实战演练

(1) 使用下列命令将一个文件设置为不可修改：

```
# chattr +i file
```

(2) 这样文件file就变为了不可修改状态。来试试下面的命令:

```
rm file
rm: cannot remove 'file': Operation not permitted
```

(3) 如果需要使文件恢复可写状态，移除不可修改属性即可：

```
chattr -i file
```

3.7　批量生成空白文件

有时候我们可能需要为程序生成测试样本，这些程序面对的是上千个文件。那该怎样生成这些测试样本呢?

3.7.1　预备知识

touch命令可以用来生成空白文件或是修改文件的时间戳（如果文件已存在）。让我们来看看该命令的用法。

3.7.2　实战演练

(1) 用下面的命令创建一个名为filename的空白文件：

```
$ touch filename
```

(2) 批量生成不同名字的空白文件：

```
for name in {1..100}.txt
do
  touch $name
done
```

在上面的代码中，{1..100}会扩展成一个字符串"1, 2, 3, 4, 5, 6, 7，…100"。除了{1..100}.txt，我们还可以使用其他简写样式，比如 test{1..200}.c、test{a..z}.txt等。

如果文件已经存在，那么touch命令会将与该文件相关的所有时间戳都更改为当前时间。如果我们只想更改某些时间戳，则可以使用下面的选项。

- touch -a 只更改文件访问时间。
- touch -m 只更改文件内容修改时间。

(3) 除了将时间戳更改为当前时间，我们还能够为时间戳指定特定的时间和日期：

```
$ touch -d "Fri Jun 25 20:50:14 IST 1999" filename
```

-d使用的日期串不需要是同样的格式。它可以接受任何简短的日期格式。我们可以忽略具体时间，使用"Jan 20 2010"这种方便的日期格式。

3

3.8　查找符号链接及其指向目标

符号链接在类Unix系统中很常见。使用它的理由有很多，要么是为了便于存取，要么是为了维护相同代码库或程序的不同版本。这则攻略中我们讨论了处理符号链接的一些基本方法。

符号链接只不过是指向其他文件的指针。它在功能上类似于Mac OS中的别名或Windows中的快捷方式。删除符号链接不会影响到原始文件。

3.8.1　实战演练

我们可以按照下面的步骤来处理符号链接。

(1) 创建符号链接：

```
$ ln -s target symbolic_link_name
```

例如：

```
$ ln -l -s /var/www/ ~/web
```

这个命令在已登录用户的home目录中创建了一个名为Web的符号链接。该链接指向/var/www。

(2) 使用下面的命令来验证是否创建链接：

```
$ ls -l web
lrwxrwxrwx 1 slynux slynux 8 2010-06-25 21:34 web -> /var/www
```

web -> /var/www表明web指向 /var/www。

(3) 打印出当前目录下的符号链接：

```
$ ls -l | grep "^l"
```

(4) 使用find打印当前目录以及子目录下的符号链接：

```
$ find . -type l -print
```

(5) 使用readlink打印出符号链接所指向的目标路径：

```
$ readlink web
/var/www
```

3.8.2　工作原理

在查找当前目录下的符号链接时，grep对ls -l的输出进行过滤，使用^显示那些以l起始的行，^是字符串的起始标记。该方法利用了这样一个事实：每个符号链接的权限标记块（lrwxrwxrwx）均以字母l起始。

在使用find时，我们将type的参数指定为l，告诉find命令只搜索符号链接文件。-print选项将符号链接列表打印到标准输出（stdout）。“·”表示从当前目录开始搜索。

3.9　列举文件类型统计信息

文件类型可谓数量繁多。如果编写一个脚本，使它能够遍历目录中所有的文件，并生成一份关于文件类型细节以及每种文件类型数量的报告，这肯定很有意思。这则攻略正是教你练习如何编写一个能够遍历大量文件并收集相关细节的脚本。

3.9.1　预备知识

find命令可以通过查看文件内容来找出特定类型的文件。在Unix/Linux系统中，文件类型并不是由文件扩展名决定的（在微软Windows平台中是这么做的）。编写这个脚本的目的是从多个文件中收集文件类型统计信息。我们用关联数组存储相同类型的文件数量，用find命令获取每一个文件的类型细节。

3.9.2　实战演练

(1) 用下面的命令打印文件类型信息：

```
$ file filename
```

```
$ file /etc/passwd
/etc/passwd: ASCII text
```

(2) 打印不包括文件名在内的文件类型信息：

```
$ file -b filename
ASCII text
```

(3) 生成文件统计信息的脚本如下：

```
# !/bin/bash
# 文件名: filestat.sh

if [ $# -ne 1 ];
then
  echo "Usage is $0 basepath";
  exit
fi
path=$1

declare -A statarray;

while read line;
do
  ftype=`file -b "$line" | cut -d, -f1`
  let statarray["$ftype"]++;

done < <(find $path -type f -print)

echo ============ File types and counts =============
for ftype in "${!statarray[@]}";
do
  echo $ftype :  ${statarray["$ftype"]}
done
```

(4) 用法如下：

```
$ ./filestat.sh /home/slynux/temp
```

输出如下：

```
$ ./filetype.sh /home/slynux/programs
============ File types and counts =============
Vim swap file : 1
ELF 32-bit LSB executable : 6
ASCII text : 2
ASCII C program text : 10
```

3

3.9.3　工作原理

在脚本中声明了一个关联数组statarray，这样可以用文件类型作为数组索引，将每种文件类型的数量存入数组。每次遇到一个文件类型，就用let增加计数。find命令以递归的方式获取文件路径列表。脚本中的ftype='file -b "$line"'使用file命令获得文件类型信息。选项-b告诉file命令只打印文件类型（不包括文件名）。输出的文件类型信息包含很多细节，比如图像编码以及分辨率（如果是图像文件的话）。对于这些细节我们并不感兴趣，我们只需要基本的信息就够了。各种细节信息是由逗号分隔的，例如：

```
$ file a.out -b
ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked (uses
shared libs), for GNU/Linux 2.6.15, not stripped
```

我们只需要从上面这些细节中提取ELF 32-bit LSB executable。因此我们使用cut -d, -f1，指明以逗号作为定界符，并且只打印第一个字段。

done<<(find $path -type f -print);是一段很重要的代码。它的执行逻辑如下：

```
while read line;
do something
done < filename
```

我们不用filename，而是用find命令的输出。

<(find $path -type f -print)等同于文件名。只不过它用子进程输出来代替文件名。注意，第一个<用于输入重定向，第二个<用于将子进程的输出装换成文件名。在两个<之间有一个空格，避免shell将其解释为<<操作符。

> 在Bash 3.*x*及更高的版本中，有一个新操作符<<<，可以让我们将字符串作为输入文件。利用这个新操作符，可以将loop循环的done语句改写成
>
> ```
> done <<< "`find $path -type f -print`"
> ```

${!statarray[@]}用于返回一个数组索引列表。

3.10　使用环回文件

环回（loopback）文件系统是Linux类系统中非常有趣的部分。我们通常是在设备上（例如磁盘分区）创建文件系统。这些存储设备能够以设备文件的形式来使用，比如 /dev/device_name。为了使用存储设备上的文件系统，我们需要将其挂载到一些被称为**挂载点**（mount point）的目录上。环回文件系统是指那些在文件中而非物理设备中创建的文件系统。我们可以将这些文件作为

文件系统挂载到挂载点上。这实际上可以让我们在物理磁盘上的文件中创建逻辑磁盘。

3.10.1　实战演练

让我们来看看如何在大小为1GB的文件中创建ext4文件系统。

(1) 下面的命令可以创建一个1GB大小的文件：

```
$ dd if=/dev/zero of=loobackfile.img bs=1G count=1
1024+0 records in
1024+0 records out
1073741824 bytes (1.1 GB) copied, 37.3155 s, 28.8 MB/s
```

你会发现创建好的文件大小超过了1GB。这是因为硬盘作为块设备，其分配存储空间时是按照块大小的整数倍来进行的。

(2) 用mkfs命令将1GB的文件格式化成ext4文件系统：

```
$ mkfs.ext4 loopbackfile.img
```

(3) 使用下面的命令检查文件系统：

```
$ file loobackfile.img
loobackfile.img: Linux rev 1.0 ext4 filesystem data,
UUID=c9d56c42-f8e6-4cbd-aeab-369d5056660a (extents) (large files) (huge files)
```

(4) 现在就可以挂载环回文件了：

```
# mkdir /mnt/loopback
# mount -o loop loopbackfile.img /mnt/loopback
```

-o loop用来挂载环回文件系统。

这实际上是一种快捷的挂载方法，我们无需手动连接任何设备。但是在内部，这个环回文件连接到了一个名为/dev/loop1或loop2的设备上。

(5) 我们也可以手动来操作：

```
# losetup /dev/loop1 loopbackfile.img
# mount /dev/loop1 /mnt/loopback
```

(6) 使用下面的方法进行卸载（umount）：

```
# umount mount_point
```

例如：

```
# umount /mnt/loopback
```

3

(7) 也可以用设备文件的路径作为umount命令的参数：

```
# umount /dev/loop1
```

> 注意，因为mount和umount都是特权命令，所以必须以root用户的身份来执行。

3.10.2　工作原理

我们首先使用dd命令创建了一个文件，准备将其作为环回文件使用。dd是一个用于复制原始数据（raw data）的通用命令。它将数据从if参数所指定的文件复制到of参数所指定的文件中。另外，我们指定dd以大小为1GB的块为单位进行复制，共复制1块，这样就创建了一个1GB的文件。/dev/zero是一个特殊的文件，如果读取这个文件，读出的内容都是0。

然后使用mkfts.ext4命令在该文件中创建ext4文件系统。我们需要文件系统才能将文件存储到磁盘或环回文件中。

最后，我们使用mount命令将环回文件挂载到**挂载点**上（在这个例子中是/mnt/loopback）。挂载点使得用户可以访问文件系统中的文件。在执行mount命令之前，应该先使用mkdir命令创建挂载点。选项-o loop用于指明使用的是环回文件。

当mount知道它使用的是环回文件时，它会自动在/dev中建立一个对应该环回文件的设备并进行挂载。如果想手动操作，可以使用losetup命令建立设备，然后使用mount命令挂载。

3.10.3　补充内容

让我们再来研究一下使用环回文件和挂载的其他用法。

1. 在环回镜像中创建分区

假设我们需要创建环回文件，然后分区并挂载其中某个分区。在这种情况下，没法使用mount -o loop。我们必须手动建立设备并挂载分区。使用下面的方法对文件（内容全部填充为0）进行分区：

```
# losetup /dev/loop1 loopback.img
# fdisk /dev/loop1
```

> fdisk是Linux系统中的标准分区工具，在http://www.tldp.org/HOWTO/Partition/fdisk_partitoning.html处可以找到一份有关如何使用fdisk创建分区的简明教程（记得将教程中的/dev/hdb换成/dev/loop1）。

在loopback.img中创建分区并挂载第一个分区：

```
# losetup -o 32256 /dev/loop2 loopback.img
```

/dev/loop2表示第一个分区，-o用来指定偏移量，32 256字节用于DOS分区方案[①]。第一个分区在硬盘上起始于32 256字节处。

2. 快速挂载带有分区的换回磁盘镜像

如果我们希望挂载环回磁盘镜像中的分区，可以将分区偏移量以手动的形式传递给losetup。不过，有一个更快的方法可以挂载镜像中的所有分区——kpartx。它并没有安装默认在系统中，你得使用软件包管理器进行安装：

3

```
# kpartx -v -a diskimage.img
add map loop0p1 (252:0): 0 114688 linear /dev/loop0 8192
add map loop0p2 (252:1): 0 15628288 linear /dev/loop0 122880
```

这条命令在磁盘镜像中的分区与/dev/mapper中的设备之间建立了映射，随后便可以挂载这些设备。下列命令可以用来挂载第一个分区：

```
# mount /dev/mapper/loop0p1 /mnt/disk1
```

当你处理完该设备上的操作后（使用umount卸载所有挂载过的分区），使用下列命令移除映射关系：

```
# kpartx -d diskimage.img
loop deleted : /dev/loop0
```

3. 将ISO文件作为环回文件挂载

ISO文件是光学存储介质的归档。我们可以采用挂载环回文件的方法，像挂载物理光盘一样挂载ISO文件。

我们甚至可以用一个非空目录作为挂载路径。那么在设备被卸载之前，这个挂载路径中包含的都是来自该设备的数据，而非目录中的原始内容。例如：

```
# mkdir /mnt/iso
# mount -o loop linux.iso /mnt/iso
```

现在就可以用/mnt/iso中的文件进行操作了。ISO是一个只读文件系统。

4. 使用sync即刻应用更改

当对挂载设备作出更改之后，这些改变并不会被立即写入物理设备。只有当缓冲区被写满之

① losetup 中的 -o 32256 (512*63=32256)用于设置数据偏移。由于历史原因，硬盘第一个扇区（512 字节）作为 MBR（Master Boot Record，主引导记录），其后的62个扇区作为保留扇区。

后才会进行设备回写。但是我们可以用sync命令强制将更改即刻写入：

```
$ sync
```

3.11　生成 ISO 文件及混合型 ISO

ISO镜像是一种存档格式，它存储了如CD-ROM、DVD-ROM等光盘的精确镜像。ISO镜像通常用于存储待刻录的数据。这节，我们会看到如何使用光盘来创建ISO镜像。很多人都是依赖第三方工具来创建ISO镜像。其实若使用命令行，会更简单。

我们同样需要区分可引导光盘与不可引导光盘之间的差别。可引导光盘自身具备引导能力，也可以运行操作系统或其他软件。不可引导光盘则做不到这些。很重要的一点是：将可引导光盘中的内容复制到另一张光盘上并不足以生成一张新的可引导光盘。要想保留光盘的可引导性，应该使用ISO文件将其保存为磁盘镜像。

现在，多数人会用闪存或硬盘作为光盘的代替品。当我们将一个可引导的ISO文件写入闪存后，它却再也没法引导了，除非我们使用一种专门设计用于闪存设备的混合ISO镜像。

这则攻略将带你认识ISO镜像及其处理方法。

3.11.1　预备知识

我们已经提到过多次，Unix将一切都作为文件来处理。所有的设备都是文件。那应该怎样复制设备的精确镜像呢？需要读出所有的数据，并将其写入另外一个文件，对吧？

如我们所知，cat命令可以用来读取任何数据，重定向可以用来写入文件。

3.11.2　实战演练

用下面的命令从/dev/cdrom创建一个ISO镜像：

```
# cat /dev/cdrom > image.iso
```

尽管可以奏效。但创建ISO镜像最好的方法还是使用dd工具：

```
# dd if=/dev/cdrom of=image.iso
```

mkisofs命令用于创建ISO文件系统。可以用cdrecord之类的工具将mkisofs的输出文件直接刻录到CD-ROM或DVD-ROM上。我们可以将需要的所有文件放入同一个目录中，然后用mkisofs将整个目录的内容写入一个ISO文件。方法如下：

```
$ mkisofs -V "Label" -o image.iso source_dir/
```

其中选项 -o指定了ISO文件的路径。source_dir是作为ISO文件内容来源的目录路径，选项 -V指定了ISO文件的卷标。

3.11.3 补充内容

让我们继续学习一些ISO文件相关的命令和技巧。

1. 能够启动闪存或硬盘的混合型ISO

通常无法通过将可引导的ISO文件写入USB设备来启动操作系统。但是有一种被称为“混合ISO”的特殊ISO文件可以做到这一切。

我们可以用isohybrid命令把标准ISO文件转换成混合ISO。isohybrid是一个比较新的工具，大多数的Linux发行版中还未包含这个工具。你可以从http://syslinux.zytor.com下载syslinux软件包。

来看看下面的命令：

```
# isohybrid image.iso
```

执行这个命令，我们将获得一个名为image.iso的混合ISO，它可用于写入USB存储设备。

将该ISO写入USB存储设备：

```
# dd if=image.iso of=/dev/sdb1
```

你可以用适当的设备代替/dev/sdb1，或者使用cat命令：

```
# cat image.iso >> /dev/sdb1
```

2. 用命令行刻录ISO

cdrecord命令可以用来将ISO文件刻入CD-ROM或DVD-ROM。刻录CD-ROM的方法如下：

```
# cdrecord -v dev=/dev/cdrom image.iso
```

还有一些其他的选项，如下所示:

- 我们可以用-speed选项指定刻录速度：

  ```
  -speed SPEED
  ```

  例如：

  ```
  # cdrecord -v dev=/dev/cdrom image.iso -speed 8
  ```

  参数8表明其刻录速度为8x。

- 刻录CD-ROM时也可以采用多区段（multisession）方式，这样就能在一张光盘上分多次

刻录数据。多区段刻录需要使用 `-multi`选项：

```
# cdrecord -v dev=/dev/cdrom image.iso -multi
```

3. 玩转CD-ROM托盘

如果你是用的是桌面电脑，不妨试试下面的命令来找点乐趣。

- **`$ eject`**

  这个命令可以弹出光驱托盘。

- **`$ eject -t`**

  这个命令可以合上光驱托盘。

  不妨试着写一个可以让托盘重复开合*n*次的循环吧。

3.12　查找文件差异并进行修补

当一个文件有多个版本时，如果能够重点标记出这些版本之间的不同而无须通过人工查看来比较，那就简直是太棒了。要是文件很大，光靠人工进行比较可是件极其费时费力的活儿。这则攻略为你演示如何用行号对文件之间的差异进行重点标记。当多名开发人员在文件繁多的环境下工作时，如果某个人对其中某个文件进行了修改，那么应该将这个修改告知其他所有开发人员。如果发送整个源代码，不仅浪费磁盘存储空间，而且单靠手动检查这些修改就很耗时间。这时，发送一个差异文件就显得很有用了。它只包含那些被修改过、添加或删除的行以及行号。这个差异文件被称为**修补文件**（patch file）。我们可以用patch命令将修补文件中包含的更改信息应用到原始文件。也可以再次进行修补来撤销改变。来看看如何才能实现这一切。

3.12.1　实战演练

`diff`命令可以生成差异文件。

(1) 为了生成差异信息，先创建下列文件。

- **文件 1**：version1.txt

```
this is the original text
line2
line3
line4
happy hacking !
```

- 文件 2：version2.txt

```
this is the original text
line2
line4
happy hacking !
GNU is not UNIX
```

(2) 非一体化（nonunified）形式的diff输出（不使用 -u选项）如下：

```
$ diff version1.txt version2.txt
3d2
<line3
6c5
> GNU is not UNIX
```


(3) 一体化形式的diff输出如下：

```
$ diff -u version1.txt version2.txt
--- version1.txt  2010-06-27 10:26:54.384884455 +0530
+++ version2.txt  2010-06-27 10:27:28.782140889 +0530
@@ -1,5 +1,5 @@
this is the original text
line2
-line3
line4
happy hacking !
-
+GNU is not UNIX
```

选项-u用于生成一体化输出。因为一体化输出的可读性更好，更易于看出两个文件之间的差异，所以人们往往更喜欢这种输出形式。

在一体化diff输出中，以 + 起始的是新加入的行，以 - 起始的是删除的行。

(4) 修补文件可以通过将diff的输出重定向到一个文件来生成：

```
$ diff -u version1.txt version2.txt > version.patch
```

现在就可以用patch命令将修改应用于任意文件。当应用于version1.txt时，我们就可以得到version2.txt；而当应用于version2.txt时，就可以得到version1.txt。

(5) 用下列命令来进行修补：

```
$ patch -p1 version1.txt < version.patch
patching file version1.txt
```

version1.txt的内容现在和verson2.txt的内容一模一样。

(6) 下面的命令可以撤销做出的修改：

```
$ patch -p1 version1.txt < version.patch
patching file version1.txt
Reversed (or previously applied) patch detected!  Assume -R? [n] y
# 修改被撤销
```

如上例所示，修补已修补的文件将撤销修改。

在撤销修改时，若使用patch命令的 -R选项，则不会提示用户y/n。

3.12.2　补充内容

让我们再看一些diff的其他特性。

生成目录的差异信息

diff命令也能够以递归的形式作用于目录。它会对目录中的所有内容生成差异输出。使用下面的命令：

```
$ diff -Naur directory1 directory2
```

上面命令中出现的选项含义如下。

- -N：将所有缺失的文件视为空文件。
- -a：将所有文件视为文本文件。
- -u：生成一体化输出。
- -r：遍历目录下的所有文件。

3.13　使用 **head** 与 **tail** 打印文件的前 10 行和后 10 行

当查看上千行的大文件时，我们可不会用cat命令把整个文件内容给打印出来。相反，我们只会查看文件的一小部分内容（例如文件的前10行或后10行）。有时候可能需要打印出文件的前*n*行或后*n*行，也有可能需要打印出除了前*n*行或后*n*行之外所有的行。

还有一种情况是打印文件的第*m*行至第*n*行。

head和tail命令可以帮助我们实现这些需求。

实战演练

head命令总是读取输入文件的头部。

(1) 打印前10行：

```
$ head file
```

(2) 从stdin读取数据：

```
$ cat text | head
```

(3) 指定打印前几行：

```
$ head -n 4 file
```

该命令会打印文件的前4行。

(4) 打印除了最后*M*行之外所有的行：

```
$ head -n -M file
```

> 注意，-M表示一个负数，并非选项。

例如，用下面的代码打印除了最后5行之外的所有行：

```
$ seq 11 | head -n -5
1
2
3
4
5
6
```

而下面的命令会打印出文件的第1行至第5行：

```
$ seq 100 | head -n 5
```

(5) 将最后几行排除在打印范围之外是head非常重要的一种用法。来看看如何打印文件最后的若干行。打印文件的最后10行：

```
$ tail file
```

(6) 可以用下面的代码从stdin中读取输入：

```
$ cat text | tail
```

(7) 打印最后5行：

```
$ tail -n 5 file
```

(8) 打印除了前*M*行之外所有的行：

```
$ tail -n +(M+1)
```

例如，打印除前5行之外的所有行，M+1=6，因此使用下列命令：

```
$ seq 100 | tail -n +6
```

这条命令将打印出第6行至第100行。

tail命令的一个重要用法是从一个内容不断增加的文件中读取数据。新增加的内容总是被添加到文件的尾部，因此当新内容被写入文件的时候，可以用tail将其显示出来。只是如果简单地使用tail的话，它只会读取文件的最后10行，然后退出。但那时新的内容也许又已经被其他进程追加到文件中了。为了能够不间断地监视文件的增长，tail有一个特殊的选项 -f或--follow，它们会使tail密切关注文件中新添加的内容，并随着数据的增加持续保持更新：

```
$ tail -f growing_file
```

你可能希望将其用于日志文件。监视文件内容增加的命令如下：

```
# tail -f /var/log/messages
```

或者

```
$ dmesg | tail -f
```

我们经常会运行dmeg 查看内核的环形缓冲区消息，要么是调试USB设备，要么是查看sdX（X是对应于SCSI磁盘的sd设备的次序列号）。tail -f也可以加入一个睡眠间隔 -s，这样我们就可以设置监视文件更新的时间间隔。

tail有一个很有意思的特性：当某个给定进程结束之后，tail也会随之终结。

假设我们正在读取一个不断增长的文件，进程Foo一直在向该文件追加数据，那么tail -f就会一直执行到进程Foo结束。

```
$ PID=$(pidof Foo)
$ tail -f file --pid $PID
```

当进程Foo结束之后，tail也会跟着结束。

让我们实际演练一下。

(1) 创建一个新文件file.txt，用gedit打开这个文件（你也可以使用其他文本编辑器）。
(2) 向文件添加新内容并不断地在gedit中保存。
(3) 现在运行下列命令：

```
$ PID=$(pidof gedit)
$ tail -f file.txt --pid $PID
```

当你不断对文件进行更新时，更新的内容都会被tail命令写入终端。当关闭gedit时，tail命令也会随之结束。

3.14　只列出目录的各种方法

用脚本只列出目录似乎不是件容易事。这则攻略很值得学习，因为它介绍了多种只列出目录的方法与技巧。

3.14.1　预备知识

有很多种方法可以只列出目录。当你向其他人询问这些技术的时候，你得到的第一个答案可能是dir。但dir只不过是一个类似于ls且选项更少的命令。让我们来看看如何列出目录。

3.14.2　实战演练

有好几种方法可以列出当前路径下的目录。

(1) 使用ls -d：

```
$ ls -d */
```

(2) 使用grep结合ls -F：

```
$ ls -F | grep "/$"
```

(3) 使用grep结合ls -l：

```
$ ls -l | grep "^d"
```

(4) 使用find：

```
$ find . -type d -maxdepth 1 -print
```

3.14.3　工作原理

当使用-F时，所有的输出项都会添加上一个代表文件类型的字符，如@、*、|等。目录对应的是 / 字符。我们用grep只过滤那些以 /$ 作为行尾标记的输出项。

ls -d 输出的每一行的首字符表示文件类型。目录的文件类型字符是"d"。因此我们用grep过滤以"d"起始的行。^ 是行首标记。

find命令可以指定type的参数为目录并将maxdepth设置成1，这是因为我们不需要继续向下搜索。

3.15　在命令行中使用 pushd 和 popd 进行快速定位

当在终端或shell提示符下涉及多处位置时，我们通常所做的就是复制并粘贴路径。但如果没有GUI，只能通过命令行进行访问时，这招就难有用武之地了。举例来说，假如同时涉及/var/www、/home/slynux和 /usr/src目录，当要在这些位置之间进行切换时，每次都要通过键盘输入路径，这实在是一件很麻烦的事。此时，我们就可以使用诸如pushd和popd这种基于**命令行接口**（Command-Line Interface，CLI）的定位技术。让我们来看看它们的使用方法。

3.15.1　预备知识

pushd和popd可以用于在多个目录之间进行切换而无需复制并粘贴目录路径。pushd和popd以栈的方式来运作。我们都知道栈是一个**后进先出**（Last In First Out, LIFO）的数据结构。目录路径被存储在栈中，然后用push和pop操作在目录之间进行切换。

3.15.2　实战演练

使用pushd和popd时，可以无视cd命令。

(1) 压入并切换路径：

```
~ $ pushd /var/www
```

现在，栈中包含了 /var/www ~，当前目录切换到 /var/www。

(2) 再压入下一个目录路径：

```
/var/www $ pushd /usr/src
```

现在栈中包含 /usr/src /var/www ~，当前目录为 /usr/src。

(3) 用下面的命令查看栈内容：

```
$ dirs
/usr/src /var/www ~ /usr/share /etc
0        1        2 3          4
```

(4) 当你想切换到列表中任意一个路径时，将每条路径从0到n进行编号，然后使用你希望切换到的路径编号，例如：

```
$ pushd +3
```

这条命令会将栈进行翻转并切换到目录 /use/share。

pushd总是将路径添加到栈。如果要从栈中删除路径，可以使用popd。

(5) 要删除最后添加的路径并把当前目录更改为上一级目录，可以使用以下命令：

```
$ popd
```

假设现在栈包含 /usr/src /var/www ~ /usr/share /etc，当前目录是 /usr/share，popd会将栈更改为 /var/www ~ /usr/share /etc，并且把目录切换到/var/www。

(6) 用popd +num可以从列表中移除特定的路径。

num是从左到右，从0到*n*开始计数的。

3.15.3　补充内容

让我们再进行一些基本的目录定位练习。

在常用的目录之间切换

当涉及3个以上的目录时，可以使用pushd和popd。但是如果只涉及两个位置的时候，还有另一个更简便的方法：cd -。

如果当前路径是 /var/www，执行下面的命令：

```
/var/www $  cd /usr/src
/usr/src $ # 做点什么
```

现在要切换回 /var/www，你不需要再输入一次，只需要执行：

```
/usr/src $ cd -
```

你还可以再切换到 /usr/src：

```
/var/www $ cd -
```

3.16　统计文件的行数、单词数和字符数

统计文件的行数、单词数和字符数在文本处理工作中发挥着重要作用。很多时候，单词或字符计数被作为一种间接的技巧来生成所需要的输出样式及结果。本书在其他章节就包含了一些这样的实例。对开发人员来说，**统计代码行数**是一件重要的工作。我们可能需要对无关文件之外的特定类型的文件进行统计。将wc结合其他命令有助于我们实现这些需求。

wc是一个用于统计的工具。它是Word Count（单词统计）的缩写。来看看如何使用wc统计文件的行数、单词数和字符数。

实战演练

我们可以使用wc的各种选项来统计行数、单词数和字符数。

(1) 统计行数：

```
$ wc -l file
```

(2) 如果需要将stdin作为输入，使用下列命令：

```
$ cat file | wc -l
```

(3) 统计单词数：

```
$ wc -w file
$ cat file | wc -w
```

(4) 统计字符数：

```
$ wc -c file
$ cat file | wc -c
```

例如，我们可以按照下面的方法统计文本中的字符数：

```
echo -n 1234 | wc -c
4
```

-n用于避免echo添加额外的换行符。

(5) 当不使用任何选项执行wc时：

```
$ wc file
   1435   15763  112200
```

它会分别打印出文件的行数、单词数和字符数。

(6) 使用-L选项打印出文件中最长一行的长度：

```
$ wc file -L
205
```

3.17　打印目录树

在准备教程和文档时，将目录和文件系统以图形化的树状层次结构描述，一定会为你增色不少。在编写一些监控脚本时，用清晰易懂的树状描述方式来查看文件系统，同样颇有益处。让我们来看看这是如何实现的。

3.17.1　预备知识

tree命令以图形化的树状结构打印文件和目录。在Linux发行版中通常并没有包含这个命令。你需要用包管理器自行安装。

3.17.2　实战演练

下面是树状Unix文件系统的一个示例：

```
$ tree ~/unixfs
unixfs/
|-- bin
|   |-- cat
|   `-- ls
|-- etc
|   `-- passwd
|-- home
|   |-- pactpub
|   |   |-- automate.sh
|   |   `-- schedule
|   `-- slynux
|-- opt
|-- tmp
`-- usr
8 directories, 5 files
```

tree命令有很多有意思的选项，让我们认识一下其中的几项。

- 重点标记出匹配某种样式的文件：

  ```
  $ tree path -P PATTERN # 用通配符描述样式
  ```

  例如：

  ```
  $ tree PATH -P "*.sh" # 用一个目录路径代替PATH
  |-- home
  |   |-- packtpub
  |   |   `-- automate.sh
  ```

- 重点标记出除符合某种样式之外的那些文件：

  ```
  $ tree path -I PATTERN
  ```

- 使用 -h选项同时打印出文件和目录的大小：

  ```
  $ tree -h
  ```

3.17.3　补充内容

请看一个有趣的`tree`命令选项。

以HTML形式输出目录树

`tree`命令可以生成HTML输出。例如，用下面的命令可以创建一个包含目录树输出的HTML文件：

```
$ tree PATH -H http://localhost -o out.html
```

将http://localhost替换为适合存放输出文件的URL。将`PATH`替换为主目录的真正路径。当前目录可以用 . 作为`PATH`。

根据目录列表生成的Web页面形式如图3-1所示。

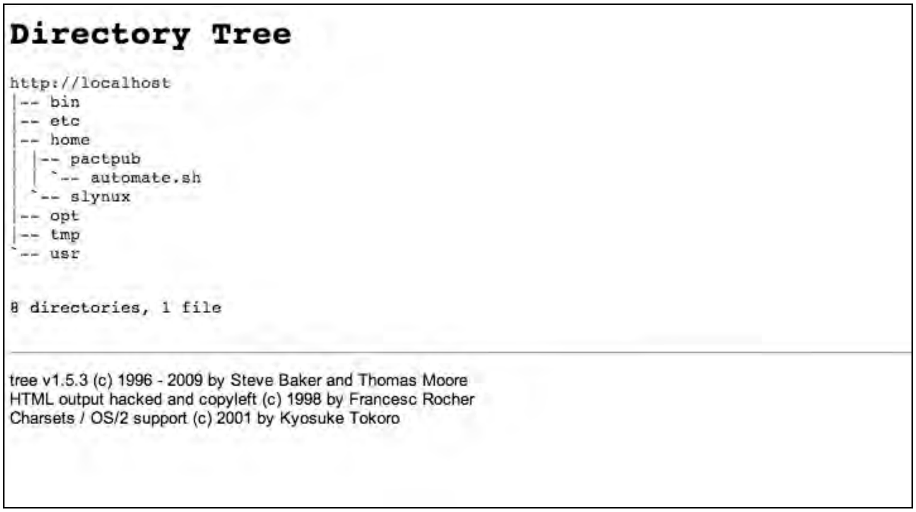


图　3-1

第 4 章

让文本飞

4

本章内容

- 使用正则表达式
- 用grep在文件中搜索文本
- 用cut按列切分文件
- 使用sed进行文本替换
- 使用awk进行高级文本处理
- 统计特定文件中的词频
- 压缩或解压缩JavaScript
- 按列合并文件
- 打印文件或行中的第*n*个单词或列
- 打印不同行或样式之间的文本
- 以逆序形式打印行
- 解析文本中的电子邮件地址和URL
- 在文件中移除包含某个单词的句子
- 对目录中的所有文件进行文本替换
- 文本切片与参数操作

4.1 简介

shell脚本语言包含了众多用于解决Unix/Linux系统问题必不可少的组件。文本处理是shell脚本擅长的重要领域之一。它可以与`sed`、`awk`、`grep`、`cut`这类优美的工具组合在一起来解决文本处理相关的问题。

有各种各样的工具能够帮助我们从不同的细节层面上处理文件，比如字符、行、单词、行列等，允许我们用多种方法来处理文本文件。正则表达式是模式匹配技术的核心。借助适合的正则表达式，可以生成我们所需的各类输出结果，例如过滤、剥离（strip）、替换、搜索等。

本章包括一系列攻略，探讨了多个与文本处理相关的问题，了解这些问题在编写用于解决实际问题的脚本时大有裨益。

4.2 使用正则表达式

正则表达式是基于模式匹配的文本处理技术的关键所在。想要在编写文本处理工具方面驾轻

就熟，你就必须对正则表达式有一个基本的理解。通配符能够匹配的文本范围相当有限。正则表达式是一种用于文本匹配的形式小巧、具有高度针对性的编程语言。`[a-z0-9_]+@[a-z0-9]+\.[a-z]+`就是一个能够匹配电子邮件地址的正则表达式。

看起来有点怪异是吧？别担心，跟着这则攻略学习，你就会发现它其实很简单。

4.2.1　实战演练

正则表达式是由字面文本和具有特殊意义的符号组成的。我们可以根据具体需求，使用它们构造出适合的正则表达式来匹配文本。因为正则表达式是一种匹配文本的通用语言，因此在这则攻略中我们不再介绍其他工具，留待本章其他攻略中再叙。

来看几个文本匹配的例子。

- 要匹配给定文本中的所有单词，可以使用下面的正则表达式：

  ```
  ( ?[a-zA-Z]+ ?)
  ```

  “?”用于匹配单词前后可能出现的空格。`[a-zA-Z]+`代表一个或多个字母（a~z和A~Z）。

- 要匹配一个IP地址，可以使用下面的正则表达式：

  ```
  [0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}
  ```

  或者

  ```
  [[:digit:]]{1,3}\.[[:digit:]]{1,3}\.[[:digit:]]{1,3}\.[[:digit:]]{1,3}
  ```

  我们知道IP地址通常的书写形式是192.168.0.2，它是由点号分割的4个整数（每一个整数的取值范围从0~255）。

  `[0-9]`或`[:digit:]`匹配数字0~9。`{1,3}`匹配1到3个数字，`\.`匹配"."。

> 这个正则表达式可以匹配所处理文本中的IP地址，但它并不检查地址的合法性。例如，形如123.300.1.1的IP地址可以被正则表达式匹配，但这却是一个非法的IP地址。不过在解析文本流时，通常目标仅仅是找出IP地址而已。

4.2.2　工作原理

先来看一看正则表达式的基本组成部分，如表4-1所示。

表　4-1

<table>
<tr><th>正则表达式</th><th>描　　述</th><th>示　　例</th></tr>
<tr><td>^</td><td>行起始标记</td><td>^tux 匹配以tux起始的行</td></tr>
<tr><td>$</td><td>行尾标记</td><td>tux$ 匹配以tux结尾的行</td></tr>
<tr><td>.</td><td>匹配任意一个字符</td><td>Hack.匹配Hackl和Hacki，但是不能匹配Hackl2和Hackil，它只能匹配单个字符</td></tr>
<tr><td>[]</td><td>匹配包含在 [字符] 之中的任意一个字符</td><td>coo[kl] 匹配cook或cool</td></tr>
<tr><td>[^]</td><td>匹配除 [^字符] 之外的任意一个字符</td><td>9[^01] 匹配92、93，但是不匹配91或90</td></tr>
<tr><td>[-]</td><td>匹配 [] 中指定范围内的任意一个字符</td><td>[1-5] 匹配从1～5的任意一个数字</td></tr>
<tr><td>?</td><td>匹配之前的项1次或0次</td><td>colou?r 匹配color或colour，但是不能匹配colouur</td></tr>
<tr><td>+</td><td>匹配之前的项1次或多次</td><td>Rollno-9+ 匹配Rollno-99、Rollno-9，但是不能匹配Rollno-</td></tr>
<tr><td>*</td><td>匹配之前的项0次或多次</td><td>co*l 匹配cl、col、coool等</td></tr>
<tr><td>()</td><td>创建一个用于匹配的子串</td><td>ma(tri)?x 匹配max或maxtrix</td></tr>
<tr><td>{n}</td><td>匹配之前的项n次</td><td>[0-9]{3} 匹配任意一个三位数，[0-9]{3} 可以扩展为[0-9][0-9][0-9]</td></tr>
<tr><td>{n,}</td><td>之前的项至少需要匹配n次</td><td>[0-9]{2,} 匹配任意一个两位或更多位的数字</td></tr>
<tr><td>{n,m}</td><td>指定之前的项所必需匹配的最小次数和最大次数</td><td>[0-9]{2,5} 匹配从两位数到五位数之间的任意一个数字</td></tr>
<tr><td>|</td><td>交替——匹配 | 两边的任意一项</td><td>Oct (1st | 2nd) 匹配Oct 1st或Oct 2nd</td></tr>
<tr><td>\</td><td>转义符可以将上面介绍的特殊字符进行转义</td><td>a\.b匹配a.b，但不能匹配ajb。通过在 . 之间加上前缀 \ ，从而忽略了 . 的特殊意义</td></tr>
</table>

更多细节请参考：http://www.linuxforu.com/2011/04/sed-explained-part-1/。

4.2.3　补充内容

下面就看一看如何在正则表达式中赋予某些字符特殊含义。

1. 处理特殊字符

正则表达式用$、^、.、*、+、{以及}等作为特殊字符。但是如果我们希望将这些字符作为非特殊字符（表示普通字面含义的字符），应该怎么做呢？来看一个正则表达式的例子：a.txt。

它会匹配字符a，然后是任意字符，接着是字符串txt。但是我们希望'.'能够匹配字面意义上的'.'，而非任意字符。所以我们在这个字符之前加上了一个反斜杠\（这叫做“将该字符进行转义”）。这表明正则表达式希望匹配的是字面字符，而不是它所代表的特殊含义。因此，最终的正则表达式就变成了a\.txt。

2. 可视化正则表达式

正则表达式有时很难理解，但是人们更易于理解带有图示的事物，因此就出现了一些将正则表达式进行可视化的工具。你可以通过http://www.regexper.com找到同类工具中的一个。你可以在这个工具中输入正则表达式，然后它会创建出一个漂亮的图示来帮助你理解。图4-1就是上一节那个正则表达式的可视化截图。

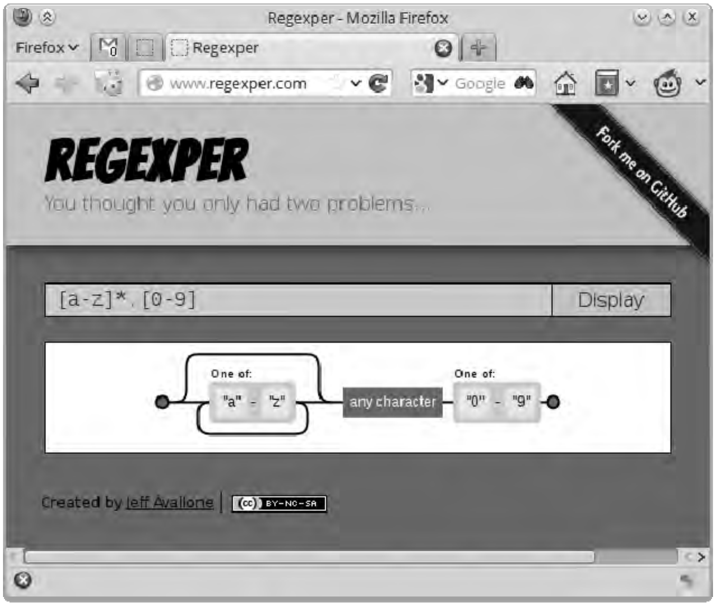


图　4-1

4.3　用 grep 在文件中搜索文本

在文件中进行搜索是文本处理中一项重要工作。我们也许需要在某些多达上千行的文件中查找所需要的数据。这则攻略将教你如何从大量数据中定位符合规格的数据。

4.3.1　实战演练

grep命令作为Unix中用于文本搜索的神奇工具，能够接受正则表达式，生成各种格式的输出。除此之外，它还有大量有趣的选项。让我们看看具体的用法。

(1) 搜索包含特定模式的文本行：

```
$ grep pattern filename
this is the line containing pattern
```

或者

```
$ grep "pattern" filename
this is the line containing pattern
```

(2) 也可以像下面这样从stdin中读取：

```
$ echo -e "this is a word\nnext line" | grep word
this is a word
```

(3) 单个grep命令也可以对多个文件进行搜索：

```
$ grep "match_text" file1 file2 file3 ...
```

(4) 用--color选项可以在输出行中着重标记出匹配到的单词：

```
$ grep word filename --color=auto
this is the line containing word
```


(5) grep命令只解释match_text中的某些特殊字符。如果要使用正则表达式，需要添加-E选项——这意味着使用扩展（extended）正则表达式。或者也可以使用默认允许正则表达式的grep命令——egrep。例如：

```
$ grep -E "[a-z]+" filename
```

或者

```
$ egrep "[a-z]+" filename
```

(6) 只输出文件中匹配到的文本部分，可以使用选项 -o：

```
$ echo this is a line. | egrep -o "[a-z]+\."
line.
```

(7) 要打印除包含match_pattern行之外的所有行，可使用：

```
$ grep -v match_pattern file
```

选项-v可以将匹配结果进行反转（invert）。

(8) 统计文件或文本中包含匹配字符串的行数：

```
$ grep -c "text" filename
10
```

需要注意的是-c只是统计匹配行的数量，并不是匹配的次数。例如：

```
$ echo -e "1 2 3 4\nhello\n5 6" | egrep  -c "[0-9]"
2
```

尽管有6个匹配项，但命令只打印出2，这是因为只有两个匹配行。在单行中出现的多次匹配只被统计为一次。

(9) 要文件中统计匹配项的数量，可以使用下面的技巧：

```
$ echo -e "1 2 3 4\nhello\n5 6" | egrep -o "[0-9]" | wc -l
6
```

(10) 打印出包含匹配字符串的行号：

```
$ cat sample1.txt
gnu is not unix
linux is fun
bash is art
$ cat sample2.txt
planetlinux

$ grep linux -n sample1.txt
2:linux is fun
```

或者

```
$ cat sample1.txt | grep linux -n
```

如果涉及多个文件，它也会随着输出结果打印出文件名：

```
$ grep linux -n sample1.txt sample2.txt
sample1.txt:2:linux is fun
sample2.txt:2:planetlinux
```

(11) 打印模式匹配所位于的字符或字节偏移：

```
$ echo gnu is not unix | grep -b -o "not"
7:not
```

一行中字符串的字符偏移是从该行的第一个字符开始计算，起始值是0。在上面的例子中，"`not`"的偏移值是7（也就是说，not是从该行的第7个字符开始的，即“gnu is not unix”这一行）。

选项 `-b`总是和 `-o`配合使用。

(12) 搜索多个文件并找出匹配文本位于哪一个文件中：

```
$ grep -l linux sample1.txt sample2.txt
sample1.txt
sample2.txt
```

和`-l`相反的选项是`-L`，它会返回一个不匹配的文件列表。

4.3.2 补充内容

我们已经看过了grep命令的基本用法。不过grep的本事可不止如此，它还有更多的特性。接着看看grep的其他选项。

1. 递归搜索文件

如果需要在多级目录中对文本进行递归搜索，可以使用：

```
$ grep "text" . -R -n
```

命令中的“.”指定了当前目录。

> grep的选项-R和-r功能一样。

例如：

```
$ cd src_dir
$ grep "test_function()" . -R -n
./miscutils/test.c:16:test_function();
```

test_function()位于miscutils/test.c的第16行。

> 这是开发人员使用最多的命令之一。它用于查找某些文本位于哪些源码文件中。

2. 忽略样式中的大小写

选项 -i可以使匹配样式不考虑字符的大小写，例如：

```
$ echo hello world | grep -i "HELLO"
hello
```

3. 用grep匹配多个样式

在进行匹配的时候通常只指定一个样式。然而，我们可以用选项 -e来指定多个匹配样式：

```
$ grep -e "pattern1" -e "pattern"
```

例如：

```
$ echo this is a line of text | grep -e "this" -e "line" -o
this
line
```

还有另一种方法也可以指定多个样式。我们可以提供一个样式文件用于读取样式。在样式文件中逐行写下需要匹配的样式，然后用选项 -f执行grep：

```
$ grep -f pattern_filesource_filename
```

例如：

```
$ cat pat_file
hello
cool

$ echo hello this is cool | grep -f pat_file
hello this is cool
```

4. 在grep搜索中指定或排除文件

grep可以在搜索过程中指定（include）或排除（exclude）某些文件。我们通过通配符来指定所include文件或exclude文件。

目录中递归搜索所有的 .c和 .cpp文件：

```
$ grep "main()" . -r  --include *.{c,cpp}
```

注意，some{string1,string2,string3}会扩展成somestring1 somestring2 somestring3。

在搜索中排除所有的README文件：

```
$ grep "main()" . -r --exclude "README"
```

如果需要排除目录，可以使用 --exclude-dir选项。

如果需要从文件中读取所需排除的文件列表，使用--exclude-from FILE。

5. 使用0值字节作为后缀的grep与xargs

xargs命令通常用于将文件名列表作为命令行参数提供给其他命令。当文件名用作命令行参数时，建议用0值字节作为文件名终止符，而非空格。因为一些文件名中会包含空格字符，一旦它被误解为终结符，那么单个文件名就会被认为是两个文件名（例如，New file.txt被解析成New和file.txt两个文件名）。这个问题可以利用0值字节后缀来避免。我们使用xargs以便从诸如grep、find中接收stdin文本。这些命令可以将带有0值字节后缀的文本输出到stdout。为了指明输入的文件名是以0值字节（\0）作为终止符，需要在xargs中使用-0。

创建测试文件：

```
$ echo "test" > file1
$ echo "cool" > file2
$ echo "test" > file3
```

在下面的命令序列中，grep使用-Z选项输出以0值字节作为终结符的文件名（\0）。xargs -0读取输入并用0值字节终结符分隔文件名：

```
$ grep "test" file* -lZ | xargs -0 rm
```

-Z通常和 -l结合使用。

6. grep的静默输出

有时候，我们并不打算查看匹配的字符串，而只是想知道是否能够成功匹配。这可以通过设置grep的静默选项（-q）来实现。在静默模式中，grep命令不会输出任何内容。它仅是运行命令，然后根据命令执行成功与否返回退出状态。

如果命令运行成功会返回0，如果失败则返回非0值。

让我们来看一个脚本，这个脚本利用grep的静默模式来测试文本匹配是否存在于某个文件中。


```
#!/bin/bash
# 文件名：silent_grep.sh
# 用途：测试文件是否包含特定的文本内容

if [ $# -ne 2 ]; then
  echo "Usage: $0 match_text filename"
  exit 1
fi

match_text=$1
filename=$2
grep -q "$match_text" $filename

if [ $? -eq 0 ]; then
  echo "The text exists in the file"
else
  echo "Text does not exist in the file"
fi
```

这个silent_grep.sh脚本使用一个用于匹配的单词（Student）和一个文件名（student_data.txt）作为命令参数：

```
$ ./silent_grep.sh Student student_data.txt
The text exists in the file
```

7. 打印出匹配文本之前或之后的行

基于上下文的打印是grep的特色之一。假设已经找到了给定文本的匹配行，通常情况下grep只会打印出这一行。但我们也许需要匹配行之前或之后的*n*行，也可能两者皆要。这可以在grep中用前后行控制选项来实现。来看看具体的做法。

要打印匹配某个结果之后的3行，使用 -A选项：

```
$ seq 10 | grep 5 -A 3
5
6
7
8
```

要打印匹配某个结果之前的3行，使用 -B选项：

```
$ seq 10 | grep 5 -B 3
2
3
4
5
```

要打印匹配某个结果之前以及之后的3行，使用-C选项：

```
$ seq 10 | grep 5 -C 3
2
3
4
5
6
7
8
```

如果有多个匹配，那么使用--作为各部分之间的定界符：

```
$ echo -e "a\nb\nc\na\nb\nc" | grep a -A 1
a
b
--
a
b
```

4.4　用 cut 按列切分文件

我们可能需要按列，而不是按行来切分文件。假设我们有一个文本文件，其中按行包含了学生的报表信息，例如Roll、Name、Mark、Percentage。我们需要将学生的姓名或者是第*n*列提取到另一个文件，也可能需要其中两列甚至更多。这则攻略将为你演示如何完成这项任务。

4.4.1　实战演练

cut是一个帮我们将文本按列进行切分的小巧工具。它也可以指定分隔每列的定界符。在cut的术语中，每列被称为一个**字段**。

(1) 要提取特定的字段或列，可以使用下面的语法：

```
cut -f FIELD_LIST filename
```

FIELD_LIST是需要显示的列。它由列号组成，彼此之间用逗号分隔。例如：

```
$ cut -f 2,3 filename
```

这条命令将显示第2列和第3列。

(2) cut也能够从stdin中读取输入文本。

制表符是字段或列的默认定界符。对于没有定界符的行，会将该行照原样打印出来。如果不想打印出这种不包含定界符的行，则可以使用cut的 -s选项。一个cut命令的例子如下：

```
$ cat student_data.txt
No  Name  Mark  Percent
1  Sarath  45  90
2  Alex  49  98
3  Anu  45  90

$ cut -f1 student_data.txt
No
1
2
3
```

(3) 提取多个字段：

```
$ cut -f2,4 student_data.txt
Name      Percent
Sarath    90
Alex      98
Anu      90
```

(4) 要打印多列，需要提供一个由逗号分隔的列号列表作为-f选项的参数。

(5) 我们也可以用 --complement选项对提取的字段进行补集运算。假设有多个字段，你希望打印出除第3列之外的所有列，则可以使用：

```
$ cut -f3 --complement student_data.txt
No  Name     Percent
1   Sarath   90
2   Alex     98
3   Anu      90
```

(6) 要指定字段的定界符，使用 -d选项：

4

```
$ cat delimited_data.txt
No;Name;Mark;Percent
1;Sarath;45;90
2;Alex;49;98
3;Anu;45;90

$ cut -f2 -d";" delimited_data.txt
Name
Sarath
Alex
Anu
```

4.4.2　补充内容

cut命令有一些选项可以将一串指定的字符作为列来显示。让我们来看看这些选项。

指定字段的字符或字节范围

假设我们不依赖定界符，但需要通过将字段定义为一个字符范围（行首记为0）来进行字段提取。这种需求也可以用cut来实现。

表4-2中列出了对应的记法。

表　4-2

记　法	范　围
`N-`	从第*N*个字节，字符或字段到行尾
`N-M`	从第*N*个字节，字符或字段到第*M*个（包括第*M*个在内）字节、字符或字段
`-M`	第1个字节，字符或字段到第*M*个（包括第*M*个在内）字节、字符或字段

用上面介绍的记法，结合下列选项将字段指定为某个范围内的字节或字符：

- `-b` 表示字节；
- `-c` 表示字符；
- `-f` 用于定义字段。

例如：

```
$ cat range_fields.txt
abcdefghijklmnopqrstuvwxyz
abcdefghijklmnopqrstuvwxyz
abcdefghijklmnopqrstuvwxyz
abcdefghijklmnopqrstuvwxy
```

你可以打印第1个到第5个字符：

```
$ cut -c1-5 range_fields.txt
abcde
abcde
abcde
abcde
```

打印前2个字符：

```
$ cut range_fields.txt -c -2
ab
ab
ab
ab
```

若要用字节作为计数单位，可以将 -c替换成 -b。

在使用 -c、-f和 -b时，我们可以指定输出定界符：

```
--output-delimiter "delimiter string"
```

当用-b或-c提取多个字段时，必须使用 --output-delimiter，否则，你就没法区分不同的字段了。例如：

```
$ cut range_fields.txt -c1-3,6-9 --output-delimiter ","
abc,fghi
abc,fghi
abc,fghi
abc,fghi
```

4.5 使用 sed 进行文本替换

sed是流编辑器（stream editor）的缩写。它是文本处理中不可或缺的工具，能够配合正则表达式使用，功能不同凡响。sed命令众所周知的一个用法是进行文本替换。这则攻略包括了sed命令大部分的常用技术。

4.5.1 实战演练

(1) sed可以替换给定文本中的字符串。

```
$ sed 's/pattern/replace_string/' file
```

或者

```
$ cat file | sed 's/pattern/replace_string/'
```

该命令从stdin中读取输入。

> 如果你用的是vi编辑器，你会发现它用于替换文本的命令和sed的非常相似。

(2) 在默认情况下，sed只会打印替换后的文本。如果需要在替换的同时保存更改，可以使用-i选项，可以将替换结果应用于原文件。很多用户在进行替换之后，会借助重定向来保存文件：

```
$ sed 's/text/replace/' file >newfile
$ mv newfile file
```

其实只需要一行命令就可以搞定，例如：

```
$ sed -i 's/text/replace/' file
```

(3) 之前看到的sed命令会将每一行中第一处符合模式的内容替换掉。但是如果要替换所有内容，我们需要在命令尾部加上参数g，其方法如下：

```
$ sed 's/pattern/replace_string/g' file
```

后缀/g意味着sed会替换每一处匹配。但是有时候我们只需要从第*n*处匹配开始替换。对此，可以使用/Ng选项。

请看下面的命令：

```
$ echo thisthisthisthis | sed 's/this/THIS/2g'
thisTHISTHISTHIS

$ echo thisthisthisthis | sed 's/this/THIS/3g'
thisthisTHISTHIS

$ echo thisthisthisthis | sed 's/this/THIS/4g'
thisthisthisTHIS
```

字符/在sed中被作为定界符使用。我们可以像下面一样使用任意的定界符：

```
sed 's:text:replace:g'
sed 's|text|replace|g'
```

当定界符出现在样式内部时，我们必须用前缀\对它进行转义：

```
sed 's|te\|xt|replace|g'
```

\|是一个出现在样式内部并经过转义的定界符。

4.5.2　补充内容

sed命令包含大量可用于文本处理的选项。将这些选项以合理的次序组合，可以只用一行命

令就解决很多复杂的问题。让我们看看这些选项。

1. 移除空白行

用sed移除空白行不过是小菜一碟。空白行可以用正则表达式 ^$ 进行匹配：

```
$ sed '/^$/d' file
```

/pattern/d会移除匹配样式的行。

在空白行中，行尾标记紧随着行首标记。

2. 直接在文件中进行替换

如果将文件名传递给sed，它会将文件内容输出到stdout。如果我们想修改文件内容，可以使用-i选项：


```
$ sed 's/PATTERN/replacement/' -i filename
```

例如，使用指定的数字替换文件中所有3位数的数字：

```
$ cat sed_data.txt
11 abc 111 this 9 file contains 111 11 88 numbers 0000

$ sed -i 's/\b[0-9]\{3\}\b/NUMBER/g' sed_data.txt
$ cat sed_data.txt
11 abc NUMBER this 9 file contains NUMBER 11 88 numbers 0000
```

上面的单行命令替换了所有的3位数字。正则表达式\b[0-9]\{3\}\b用于匹配3位数字。[0-9]表示数位取值范围，也就是说从0~9。{3}表示匹配之前的字符3次。\{3\}中的\用于转义{和}。\b表示单词边界。

> 一种有益的做法是先使用不带-i选项的sed命令，以确保正则表达式没有问题，一旦结果符合要求，再加入-i选项将更改写入文件。另外，你也可以使用下列形式的sed：
>
> ```
> sed -i .bak 's/abc/def/' file
> ```
>
> 这时的sed不仅执行文件内容替换，还会创建一个名为file.bak的文件，其中包含着原始文件内容的副本。

3. 已匹配字符串标记（&）

在sed中，我们可以用 &标记匹配样式的字符串，这样就能够在替换字符串时使用已匹配的内容。

例如：

```
$ echo this is an example | sed 's/\w\+/[&]/g'
[this] [is] [an] [example]
```

正则表达式 `\w\+` 匹配每一个单词，然后我们用`[&]`替换它。`&` 对应于之前所匹配到的单词。

4. 子串匹配标记（\1）

`&` 代表匹配给定样式的字符串。但我们也可以匹配给定样式的其中一部分。来看看具体的做法。

```
$ echo this is digit 7 in a number | sed 's/digit \([0-9]\)/\1/'
this is 7 in a number
```

这条命令将`digit 7`替换为`7`。样式中匹配到的子串是`7`。`\(pattern\)`用于匹配子串。模式被包括在使用斜线转义过的`()`中。对于匹配到的第一个子串，其对应的标记是 `\1`，匹配到的第二个子串是 `\2`，往后依次类推。下面的示例中包含了多个匹配：

```
$ echo seven EIGHT | sed 's/\([a-z]\+\) \([A-Z]\+\)/\2 \1/'
EIGHT seven
```

`([a-z]\+\)`匹配第一个单词，`([A-Z]\+\)`匹配第二个单词。`\1`和`\2`用来引用它们。这种引用被称为**向后引用**（back reference）。在替换部分，它们的次序被更改为`\2 \1`，因此结果就呈现出逆序的形式。

5. 组合多个表达式

可以利用管道组合多个`sed`命令：

```
sed 'expression' | sed 'expression'
```

它等价于

```
$ sed 'expression; expression'
```

或者

```
$ sed -e 'expression' -e expression'
```

例如：

```
$ echo abc | sed 's/a/A/' | sed 's/c/C/'
AbC
$ echo abc | sed 's/a/A/;s/c/C/'
AbC
$ echo abc | sed -e 's/a/A/' -e 's/c/C/'
AbC
```

6. 引用

sed表达式通常用单引号来引用。不过也可以使用双引号。双引号会通过对表达式求值来对其进行扩展。当我们想在sed表达式中使用一些变量时，双引号就能派上用场了。

例如：

```
$ text=hello
$ echo hello world | sed "s/$text/HELLO/"
HELLO world
```

$text的求值结果是hello。

4.6 使用 awk 进行高级文本处理

awk是一款设计用于数据流的工具。它颇有玩头的原因就在于可以对列和行进行操作。awk有很多内建的功能，比如数组、函数等，这是它和C语言的相同之处。灵活性是awk最大的优势。

4.6.1 预备知识

awk脚本的结构基本如下所示：

```
awk ' BEGIN{  print "start" } pattern { commands } END{ print "end" } file
```

awk命令也可以从stdin中读取。

awk脚本通常由3部分组成。BEGIN，END和带模式匹配选项的常见语句块。这3个部分都是可选项，在脚本中可省略任意部分。

4.6.2 实战演练

以下awk脚本被包含在单引号或双引号之间：

```
awk 'BEGIN { statements } { statements } END { end statements }'
```

也可以使用：

```
awk "BEGIN { statements } { statements } END { end statements }"
```

例如：

```
$ awk 'BEGIN { i=0 } { i++ } END{ print i}' filename
```

或者

```
$ awk "BEGIN { i=0 } { i++ } END{ print i }" filename
```

4.6.3　工作原理

awk命令的工作方式如下所注。

(1) 执行BEGIN { commands }语句块中的语句。

(2) 从文件或stdin中读取一行，然后执行pattern { commands }。重复这个过程，直到文件全部被读取完毕。

(3) 当读至输入流末尾时，执行END { commands }语句块。

BEGIN语句块在awk开始从输入流中读取行之前被执行。这是一个可选的语句块，诸如变量初始化、打印输出表格的表头等语句通常都可以写入BEGIN语句块中。

END语句块和BEGIN语句块类似。END语句块在awk从输入流中读取完所有的行之后即被执行。像打印所有行的分析结果这类汇总信息，都是在END语句块中实现的常见任务（例如，在比较过所有的行之后，打印出最大数）。它也是一个可选的语句块。

最重要的部分就是pattern语句块中的通用命令。这个语句块同样是可选的。如果不提供该语句块，则默认执行{ print }，即打印所读取到的每一行。awk对于每一行，都会执行这个语句块。这就像一个用来读取行的while循环，在循环体中提供了相应的语句。

每读取一行，awk就会检查该行和提供的样式是否匹配。样式本身可以是正则表达式、条件语句以及行匹配范围等。如果当前行匹配该样式，则执行{ }中的语句。

样式是可选的。如果没有提供样式，那么awk就认为所有的行都是匹配的，并执行{ }中的语句。

让我们看看下面的例子：

```
$ echo -e "line1\nline2" | awk 'BEGIN{ print "Start" } { print } END{ print "End" } '
Start
line1
line2
End
```

当使用不带参数的print时，它会打印出当前行。关于print，需要记住两件重要的事情：当print的参数是以逗号进行分隔时，参数打印时则以空格作为定界符。在awk的print语句中，双引号是被当做拼接操作符（concatenation operator）使用的。

例如：

```
$ echo | awk '{ var1="v1"; var2="v2"; var3="v3"; \
print var1,var2,var3 ; }'
```

该语句将按照下面的格式打印变量值：

```
v1 v2 v3
```

echo命令向标准输出写入一行，因此awk的`{ }`语句块中的语句只被执行一次。如果awk的标准输入包含多行，那么`{ }`语句块中的命令就会被执行多次。

拼接的使用方法如下：

```
$ echo | awk '{ var1="v1"; var2="v2"; var3="v3"; \
print var1 "-" var2 "-" var3 ; }'
```

输出为：

```
v1-v2-v3
```

`{ }`类似于一个循环体，会对文件中的每一行进行迭代。

> 我们通常将变量初始化语句（如var=0）以及打印文件头部的语句放入BEGIN语句块中。在END{}语句块中，往往会放入打印结果等语句。

4.6.4 补充内容

awk命令具有丰富的特性。要想洞悉awk编程的精妙之处，首先应该熟悉awk重要的选项和功能。让我们来看看awk的一些重要功能。

1. 特殊变量

以下是可以用于awk的一些特殊变量。

- NR：表示记录数量，在执行过程中对应于当前行号。
- NF：表示字段数量，在执行过程中对应于当前行的字段数。
- $0：这个变量包含执行过程中当前行的文本内容。
- $1：这个变量包含第一个字段的文本内容。
- $2：这个变量包含第二个字段的文本内容。

例如：

```
$ echo -e "line1 f2 f3\nline2 f4 f5\nline3 f6 f7" | \
```

```
awk '{
print "Line no:"NR",No of fields:"NF, "$0="$0, "$1="$1,"$2="$2,"$3="$3
}'
Line no:1,No of fields:3 $0=line1 f2 f3 $1=line1 $2=f2 $3=f3
Line no:2,No of fields:3 $0=line2 f4 f5 $1=line2 $2=f4 $3=f5
Line no:3,No of fields:3 $0=line3 f6 f7 $1=line3 $2=f6 $3=f7
```

我们可以用print $NF打印一行中最后一个字段，用 $(NF-1)打印倒数第二个字段，其他字段依次类推即可。

awk的printf()函数的语法和C语言中的同名函数一样。我们也可以用这个函数来代替print。

再来看awk的一些基本用法。打印每一行的第2和第3个字段：

```
$awk '{ print $3,$2 }'  file
```

要统计文件中的行数，使用下面的命令：

```
$ awk 'END{ print NR }' file
```

这里只使用了END语句块。每读入一行，awk会将NR更新为对应的行号。当到达最后一行时，NR中的值就是最后一行的行号，于是，位于END语句块中的NR就包含了文件的行数。

你可以将每一行中第一个字段的值按照下面的方法进行累加：

```
$ seq 5 | awk 'BEGIN{ sum=0; print "Summation:" }
{ print $1"+"; sum+=$1 } END { print "=="; print sum }'
Summation:
1+
2+
3+
4+
5+
==
15
```

2. 将外部变量值传递给awk

借助选项 -v，我们可以将外部值（并非来自stdin）传递给awk：

```
$ VAR=10000
$ echo | awk -v VARIABLE=$VAR '{ print VARIABLE }'
10000
```

还有另一种灵活的方法可以将多个外部变量传递给awk，例如：

```
$ var1="Variable1" ; var2="Variable2"
$ echo | awk '{ print v1,v2 }' v1=$var1 v2=$var2
```

```
Variable1 Variable2
```

当输入来自于文件而非标准输入时，使用下列命令：

```
$ awk '{ print v1,v2 }' v1=$var1 v2=$var2 filename
```

在上面的方法中，变量之间用空格分隔，以键-值对的形式（`v1=$var1 v2=$var2`）作为awk的命名行参数紧随在BEGIN、{}和END语句块之后。

3. 用getline读取行

awk通常默认读取一个文件的所有行。如果只想读取某一行，可以使用getline函数。有时候，我们需要从BEGIN语句块中读取第一行。

语法：getline var。变量var就包含了特定行的内容。如果调用不带参数的getline，我们可以用 $0、$1和$2访问文本行的内容。

4

例如：

```
$ seq 5 | awk 'BEGIN { getline; print "Read ahead first line", $0 } { print $0 }'
Read ahead first line 1
2
3
4
5
```

4. 使用过滤模式对awk处理的行进行过滤

我们可以为需要处理的行指定一些条件，例如：

```
$ awk 'NR < 5' # 行号小于5的行
$ awk 'NR==1,NR==4' # 行号在1到5之间的行
$ awk '/linux/' # 包含样式linux的行（可以用正则表达式来指定模式）
$ awk '!/linux/' # 不包含包含模式为linux的行
```

5. 设置字段定界符

默认的字段定界符是空格。我们也可以用 -F "delimiter"明确指定一个定界符：

```
$ awk -F: '{ print $NF }' /etc/passwd
```

或者

```
awk 'BEGIN { FS=":" } { print $NF }' /etc/passwd
```

在BEGIN语句块中则可以用OFS="delimiter"设置输出字段的定界符。

6. 从awk中读取命令输出

在下面的代码中，echo会生成一个空白行。变量cmdout包含命令grep root /etc/passwd的输出，该命令会打印出包含root的行。

将命令的输出结果读入变量output的语法如下：

```
"command" | getline output ;
```

例如：

```
$ echo | awk '{ "grep root /etc/passwd" | getline cmdout ; print cmdout }'
root:x:0:0:root:/root:/bin/bash
```

通过使用getline，我们将外部shell命令的输出读入变量cmdout。

awk支持以文本作为索引的关联数组。

7. 在awk中使用循环

在awk中可以使用for循环，其格式如下：

```
for(i=0;i<10;i++) { print $i ; }
```

或者

```
for(i in array) { print array[i]; }
```

8. awk内建的字符串控制函数

awk有很多内建的字符串控制函数，让我们认识一下其中部分函数。

- length(string)：返回字符串的长度。
- index(string, search_string)：返回search_string在字符串中出现的位置。
- split(string, array, delimiter)：用定界符生成一个字符串列表，并将该列表存入数组。
- substr(string, start-position, end-position)：在字符串中用字符起止偏移量生成子串，并返回该子串。
- sub(regex, replacement_str, string)：将正则表达式匹配到的第一处内容替换成replacment_str。
- gsub(regex, replacment_str, string)：和sub()类似。不过该函数会替换正则表达式匹配到的所有内容。
- match(regex, string)：检查正则表达式是否能够匹配字符串。如果能够匹配，返回非0值；否则，返回0。match()有两个相关的特殊变量，分别是RSTART和RLENGTH。变

量RSTART包含正则表达式所匹配内容的起始位置，而变量RLENGTH包含正则表达式所匹配内容的长度。

4.7 统计特定文件中的词频

统计文件中使用单词的频率是一个不错的练习，在这里能够应用我们业已习得的文本处理技巧。词频统计的方法有很多。让我们看看具体的做法。

4.7.1 预备知识

我们可以使用关联数组、awk、sed、grep等不同的方式来解决这个问题。**单词**是由空格或句号分隔的字母组合。首先我们应该解析出给定文件中出现的所有单词，然后统计每个单词的出现的次数。单词解析可以用正则表达式配合sed、awk或grep等工具来完成。

4.7.2 实战演练

我们已经了解了实现原理。现在来动手创建如下的脚本：

```
#!/bin/bash
# 文件名：word_freq.sh
# 用途： 计算文件中单词的词频

if [ $# -ne 1 ];
then
  echo "Usage: $0 filename";
  exit -1
fi

filename=$1

egrep -o "\b[[:alpha:]]+\b" $filename | \

awk '{ count[$0]++ }
END{ printf("%-14s%s\n","Word","Count") ;
for(ind in count)
{  printf("%-14s%d\n",ind,count[ind]);  }

}'
```

输出如下：

```
$ ./word_freq.sh words.txt
Word          Count
```

```
used          1
this            2
counting   1
```

4.7.3 工作原理

egrep -o "\b[[:alpha:]]+\b" $filename只用于输出单词。用 -o选项打印出由换行符分隔的匹配字符序列。这样我们就可以在每行中列出一个单词。

\b 是单词边界标记符。[:alpha:] 是表示字母的字符类。awk命令用来避免对每一个单词进行迭代。因为awk默认会逐行执行{}块中的语句，所以我们就不需要再为同样的事编写循环了。借助关联数组，当执行count[$0]++时，单词计数就增加。最后，在END{}语句块中通过迭代所有的单词，就可以打印出单词及它们各自出现的次数。

4.7.4 参考

- 1.7节讲解了Bash中的数组。
- 4.6节介绍了awk命令。

4.8 压缩或解压缩 JavaScript

JavaScript广泛用于网站设计。在编写JavaScript代码时，出于代码可读性与方便维护方面的考虑，我们有必要使用一些空格、注释和制表符。但这些会增加JavaScript文件的大小。随着文件体积的增加，网页载入的时间也随之延长。因此，多数专业网站为了加快页面载入速度，都会对JavaScript文件进行压缩。这多是通过压缩空白字符和换行符的数量来实现的（也被称为minified JS）。对于压缩过的JavaScript，还可以通过加入足够的空白字符和换行符进行解压缩，以便恢复代码的可读性。这则攻略就尝试利用shell中实现类似的功能。

4.8.1 预备知识

我们准备写一个JavaScript压缩工具或代码混乱器，当然，还包括与之对应的解压缩工具。来考虑下面的Javascript代码：

```
$ cat sample.js
function sign_out()
{

  $("#loading").show();
  $.get("log_in",{logout:"True"},
```

```
    function(){
      window.location="";
    });
}
```

下面是压缩JavaScript所需要完成的工作：

(1) 移除换行符和制表符；
(2) 压缩空格；
(3) 替换注释 `/*内容*/`。

要解压缩或者恢复JavaScript的可读性，我们则需要：

- ❑ 用 `;\n` 替换`;`；
- ❑ 用 `{\n` 替换 `{`，`\n}` 替换`}`。

4.8.2 实战演练

按照之前叙述过的步骤，我们使用下面的命令序列：

```
$ cat sample.js |  \
tr -d '\n\t' |  tr -s ' ' \
| sed 's:/\*.*\*/::g' \
| sed 's/ \?\([{}();,:]\) \?/\1/g'
```

得到输出：

```
function sign_out(){$("#loading").show();$.get("log_in",{logout:"True"},
function(){window.location="";});}
```

接着写一个可以将这些混乱的代码恢复正常的解压缩脚本：

```
$ cat obfuscated.txt | sed 's/;/;\n/g; s/{/{\n\n/g; s/}/\n\n}/g'
```

或者

```
$ cat obfuscated.txt | sed 's/;/;\n/g' | sed 's/{/{\n\n/g' | sed 's/}/\n\n}/g'
```

> 该脚本在使用上存在局限：它会删除本不该删除的空格。假如有下列语句：
>
> ```
> var a = "hello  world"
> ```
>
> 两个空格会被转换成一个。这种问题可以使用我们讲解过的模式匹配工具来解决。如果需要处理关键JavaScript代码，最好还是使用功能完善的工具来实现。

4.8.3　工作原理

通过执行下面的步骤来进行压缩。

(1) 移除 '\n' 和 '\t'：

```
tr -d '\n\t'
```

(2) 移除多余的空格：

```
tr -s ' '或者sed 's/[ ]\+/ /g'
```

(3) 移除注释：

```
sed 's:/\*.*\*/::g'
```

因为我们需要使用 /* 和 */，所以用:作为sed的定界符，这样就不必对 / 进行转义了。

* 在 sed 中被转义为 *。

.* 用来匹配 /* 与 */ 之间所有的文本。

(4) 移除{、}、(、)、;、:以及,前后的空格。

```
sed 's/ \?\([{}();,:]\) \?/\1/g'
```

上面的sed语句含义如下。

- sed代码中的 / \?\([{ }();,:]\) \?/ 用于匹配，/\1/g 用于替换。
- \([{ }();,:]\) 用于匹配集合 [{ }( ) ; , :]（出于可读性方面的考虑，在这里加入了空格）中的任意一个字符。\(和\)是分组操作符，用于记忆所匹配的内容，以便在替换部分中进行向后引用。对(和)转义之后，它们便具备了另一种特殊的含义，进而可以将它们作为分组操作符。位于分组操作符前后的 \? 用来匹配可能出现在字符集合前后的空格。
- 在命令的替换部分，匹配字符串（也就是一个可选的空格、一个来自字符集的字符再加一个可选的空格）被匹配的子字符串所替换。对于匹配的子字符串使用了向后引用，并通过分组操作符()记录了匹配内容。可以用符号 \1向后引用分组所匹配的内容。

解压缩命令工作方式如下：

- s/;/;\n/g 将;替换为;\n;
- s/{/{\n\n/g 将 { 替换为 {\n\n;
- s/}/\n\n}/g 将 } 替换为 \n\n}。

4.8.4 参考

- 2.6节讲解了tr命令。
- 4.5节讲解了sed命令。

4.9 按列合并多个文件

很多时候我们需要按列拼接文件，比如要将每一个文件的内容作为单独的一列。而cat命令所实现的拼接通常是按照行来进行的。

4.9.1 实战演练

可以用paste命令实现按列拼接，其语法如下：

```
$ paste file1 file2 file3 …
```

让我们来尝试一下：

```
$ cat file1.txt
1
2
3
4
5
$ cat file2.txt
slynux
gnu
bash
hack
$ paste file1.txt file2.txt
1slynux
2gnu
3bash
4hack
5
```

默认的定界符是制表符，也可以用 -d明确指定定界符，例如：

```
$ paste file1.txt file2.txt -d ","
1,slynux
2,gnu
3,bash
4,hack
5,
```

4.9.2　参考

4.4节讲解了如何从文本文件中提取数据。

4.10　打印文件或行中的第 *n* 个单词或列

我们可能有一个包含了多列数据的文件，不过只有其中的一小部分能派上用场。例如在以成绩排序的学生列表中，我们希望得到成绩最高的四名学生。来看看如何实现。

4.10.1　实战演练

处理这种任务最为广泛的方法就是借助awk。同样也可以使用cut来实现。

(1) 用下面的命令打印第5列：

```
$ awk '{ print $5 }' filename
```

(2) 也可以打印多列数据，并在各列间插入指定的字符串。

如果要打印当前目录下各文件的权限和文件名，可以使用下列命令：

```
$ ls -l | awk '{ print $1 " :  " $8 }'
-rw-r--r-- :  delimited_data.txt
-rw-r--r-- :  obfuscated.txt
-rw-r--r-- :  paste1.txt
-rw-r--r-- :  paste2.txt
```

4.10.2　参考

- 4.4节讲解了如何从文本文件中提取数据。
- 4.6节讲解了awk命令。

4.11　打印行或样式之间的文本

我们有时候可能需要根据某些条件打印文本的某一部分，比如由行号或起止样式所匹配的文本范围。让我们来看看如何实现这些需求。

4.11.1　预备知识

我们可以用awk、grep和sed这类工具来根据条件打印某些文本区域。不过，我仍然觉得awk

是最简单明了的方法。下面就看看如何使用awk完成这项任务。

4.11.2 实战演练

(1) 要打印出从M行到N行这个范围内的所有文本，使用下面的语法：

```
$ awk 'NR==M, NR==N' filename
```

也可以用stdin作为输入：

```
$ cat filename | awk 'NR==M, NR==N'
```

(2) 把M和N换成具体的数字：

```
$ seq 100 | awk 'NR==4,NR==6'
4
5
6
```

(3) 要打印处于start_pattern与end_pattern之间的文本，使用下面的语法：

```
$ awk '/start_pattern/, /end _pattern/' filename
```

例如：

```
$ cat section.txt
line with pattern1
line with pattern2
line with pattern3
line end with pattern4
line with pattern5

$ awk '/pa.*3/, /end/' section.txt
line with pattern3
line end with pattern4
```

用于awk中的样式为正则表达式。

4.11.3 参考

4.6节讲解了awk命令。

4.12 以逆序形式打印行

这一节的内容很简单。讲的东西可能看起来用处不大，不过它可以用来在Bash中模拟栈结构，

所以还是有些意思的。下面我们来将一个文件中的文本行以逆序形式打印出来。

4.12.1　预备知识

只需要用点awk的小技巧就能完成这项任务。不过，命令tac可以直接搞定。这个命令的名称其实就是反过来书写的cat。

4.12.2　实战演练

先来试试tac。

(1) 该命令的语法如下：

```
tac file1 file2 …
```

它也可以从stdin中读取：

```
$ seq 5 | tac
5
4
3
2
1
```

在tac中，\n是默认的行分隔符。但我们也可以用 -s "分隔符"选项指定自己的分隔符。

(2) 使用awk的实现方式如下：

```
$ seq 9 | \
awk '{ lifo[NR]=$0 }
END{ for(lno=NR;lno>-1;lno--){ print lifo[lno]; }
}'
```

在shell脚本中，\ 可以很方便地将单行命令拆解成多行。

4.12.3　工作原理

这个awk脚本非常简单。我们将每一行都存入一个关联数组中，用行号作为数组索引（行号由NR给出），最后由awk执行END语句块。为了得到最后一行的行号，在{ }语句块中使用lno=NR。因此，这个脚本从最后一行一直迭代到第0行，将存储在数组中的各行以逆序方式打印出来。

4.13　解析文本中的电子邮件地址和 URL

从给定的文件中解析出所需要的文本是从事文本处理时常见的一项任务。诸如电子邮件地址、URL等都能够借助适合的正则表达式找出来。我们通常需要从一个包含大量无关字符及单词的电子邮件客户列表或HTML网页中将电子邮件地址解析并提取出来。

4.13.1　实战演练

能够匹配一个电子邮件地址的正则表达式如下：

```
[A-Za-z0-9._]+@[A-Za-z0-9.]+\.[a-zA-Z]{2,4}
```

例如：

```
$ cat url_email.txt
this is a line of text contains,<email> #slynux@slynux.com. </email> and email address,
blog "http://www.google.com", test@yahoo.com dfdfdfdddfdf;cool.hacks@gmail.com<br />
<a href="http://code.google.com"><h1>Heading</h1>
```

因为用到了扩展正则表达式（例如+），所以得使用egrep命令。

```
$ egrep -o '[A-Za-z0-9._]+@[A-Za-z0-9.]+\.[a-zA-Z]{2,4}'  url_email.txt
slynux@slynux.com
test@yahoo.com
cool.hacks@gmail.com
```

匹配HTTP URL的egrep正则表达式如下：

```
http://[a-zA-Z0-9\-\.]+\.[a-zA-Z]{2,4}
```

例如：

```
$ egrep -o "http://[a-zA-Z0-9.]+\.[a-zA-Z]{2,3}" url_email.txt
http://www.google.com
http://code.google.com
```

4.13.2　工作原理

如果逐个部分进行设计，这些正则表达式其实很简单。在匹配电子邮件地址的正则表达式中，我们都知道电子邮件地址可以采用name@domain.some_2-4_letter这种形式。那么，在编写正则表达式时，也要遵循同样的规则：

```
[A-Za-z0-9.]+@[A-Za-z0-9.]+\.[a-zA-Z]{2,4}
```

[A-Za-z0-9.]+ 表示在表示字面意义的字符 @ 出现之前，[] 中的字符组合应该出现一次或多次（这也正是 + 的含义）。然后 [A-Za-z0-9.] 同样应该出现一次或多次（+）。样式 \. 表示应该呈现一个表示字面意义的“.”（点号），而[a-zA-Z]{2,4}表示字母的长度应该在2到4之间（包括2和4）。

匹配HTTP URL与匹配电子邮件地址类似，只是不需要匹配name@部分。

```
http://[a-zA-Z0-9.]+\.[a-zA-Z]{2,3}
```

4.13.3　参考

- 4.2节讲解了如何使用正则表达式。
- 4.5节讲解了sed命令。

4.14　在文件中移除包含某个单词的句子

只要能写出正确的正则表达式，移除包含某个单词的句子简直就是手到擒来。这里给出了一个解决类似问题的练习。

4.14.1　预备知识

sed是进行文本替换的不二之选。这样，我们就可以通过sed用空白替代匹配的句子。

4.14.2　实战演练

先创建一个包含替换文本的文件。例如：

```
$ cat sentence.txt
Linux refers to the family of Unix-like computer operating systems that use the Linux
kernel. Linux can be installed on a wide variety of computer hardware, ranging from
mobile phones, tablet computers and video game consoles, to mainframes and supercomputers.
Linux is predominantly known for its use in servers.
```

我们的目标是移除包含“mobile phones”的句子。可以用下面的sed语句来实现：

```
$ sed 's/ [^.]*mobile phones[^.]*\.//g' sentence.txt

Linux refers to the family of Unix-like computer operating systems
that use the Linux kernel. Linux is predominantly known for its use
in servers.
```

> 这里假设没有句子分散在不同的行中。也就是说，句子总是在同一行中起止。

4.14.3　工作原理

让我们分析一下sed的正则表达式's/ [^.]*mobile phones[^.]*\.//g'。该正则表达式的格式为：'s/匹配模式/替代字符串/g'。它将与匹配样本相匹配的每一处内容都用替代字符串进行替换。

这里的匹配模式是用来匹配整句文本的正则表达式。文件中的每一句话第一个字符都是空格，句与句之间都以“.”来分隔。因此我们需要匹配内容的格式就是：空格+若干文本+需要匹配的字符串+若干文本+句点。一个句子除了作为定界符的句点之外，可以包含任意字符。因此我们要使用[^.]。[^.]*可以匹配除句点之外的任何字符的组合。用来匹配文本的“mobile phone”被放置在两个[^.]*之间。每一个匹配的句子均被//替换（注意，/与/之间没有任何内容）。

4.14.4　参考

- ❑ 4.2节讲解了如何使用正则表达式。
- ❑ 4.5节讲解了sed命令。

4.15　对目录中的所有文件进行文本替换

我们经常需要将某个目录中的所有文件内的一部分文本替换成另一部分。例如在网站的源文件目录中替换一个URI。解决这个问题最快的方法就是利用shell脚本。

4.15.1　实战演练

就目前我们所掌握的知识，首先使用find定位需要进行文本替换的文件。然后使用sed进行实际的替换操作。

假设我们希望将所有.cpp文件中的Copyright替换成Copyleft：

```
$ find . -name *.cpp -print0 |  xargs -I{} -0 sed -i 's/Copyright/Copyleft/g' {}
```

4.15.2　工作原理

我们使用find在当前目录下查找所有的.cpp文件，然后使用print0打印出以null字符（\0）

作为分隔符的文件列表。(这可以避免文件名中的空格所带来的麻烦。)接着使用通道将列表传递给xargs，后者将对应的文件作为sed的参数，由sed对文件内容进行修改。

4.15.3　补充内容

回忆一下，find有一个选项-exec，它可以对每个find查找到的文件执行命令。我们可以使用该选项实现同样的效果：

```
$ find . -name *.cpp -exec sed -i 's/Copyright/Copyleft/g' \{\} \;
```

或者

```
$ find . -name *.cpp -exec sed -i 's/Copyright/Copyleft/g' \{\} \+
```

尽管这两个命令作用相同，但第一个命令会为每个查找到的文件调用一次sed，而在第二个命令中，find会将多个文件名一并传递给sed。

4.16　文本切片及参数操作

这则攻略讲述了一些简单的文本替换技术以及Bash中可用的参数扩展简写方法。这些简单的技巧通常能够让我们免于键盘敲击之苦。

4.16.1　实战演练

让我们来练练手吧。

替换变量内容中的部分文本：

```
$ var="This is a line of text"
$ echo ${var/line/REPLACED}
This is a REPLACED of text"
```

line被替换成REPLACED。

我们可以通过指定字符串的起始位置和长度来生成子串，语法如下：

```
${variable_name:start_position:length}
```

用下面的命令可以打印第5个字符之后的内容：

```
$ string=abcdefghijklmnopqrstuvwxyz
$ echo ${string:4}
efghijklmnopqrstuvwxyz
```

从第5个字符开始，打印8个字符：

```
$ echo ${string:4:8}
efghijkl
```

起始字符的索引从0开始计数。我们也可以从后向前计数，将最后一个字符索引记为 `-1`。但如果使用负数作为索引值，则必须将负数放入括号内。`(-1)`就是最后一个字符的索引。

```
echo ${string:(-1)}
z
$ echo ${string:(-2):2}
yz
```

4.16.2 参考

4.5节讲解了从单词中切分字符。

第5章 一团乱麻？没这回事

本章内容

- Web页面下载
- 以纯文本形式下载网页
- cURL入门
- 从命令行访问Gmail
- 解析网站数据
- 图片抓取器及下载工具
- 网页相册生成器
- Twitter命令行客户端
- 基于Web后端的定义查询工具
- 查找网站中的无效链接
- 跟踪网站变更
- 以POST方式发送网页并读取响应

5.1 入门

Web正在成为反映技术发展的晴雨表和数据处理中心。尽管shell脚本没法像PHP一样在Web上大包大揽，但还是有不少活儿挺适合它。本章会研究一些用于解析网站内容、下载数据、发送数据表单以及网站维护任务自动化之类的例子。我们可以用短短几行脚本就将很多原本需要通过浏览器交互进行的活动自动化。借助命令行工具，利用HTTP协议所提供的功能，我们可以用脚本解决大部分Web自动化的问题。来尽情享受接下来的内容吧。

5.2 Web 页面下载

从给定的URL中下载文件或页面很简单。一些命令行下载工具就可以完成这项任务。

5.2.1 预备知识

`wget`是一个用于文件下载的命令行工具，选项繁多且用法灵活。

5.2.2　实战演练

用wget可以下载网页或远程文件：

```
$ wget URL
```

例如：

```
$ wget http://slynux.org
--2010-08-01 07:51:20--  http://slynux.org/
Resolving slynux.org... 174.37.207.60
Connecting to slynux.org|174.37.207.60|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 15280 (15K) [text/html]
Saving to: "index.html"

100%[======================================>] 15,280      75.3K/s   in 0.2s

2010-08-01 07:51:21 (75.3 KB/s) - "index.html" saved [15280/15280]
```

可以指定从多个URL处进行下载：

```
$ wget URL1 URL2 URL3 ..
```


5.2.3　工作原理

通常下载的文件名和URL中的文件名会保持一致，下载日志或进度被写入stdout。

你可以通过选项-O指定输出文件名。如果存在同名文件，那么会先将该同名文件清空（truncate）再写入下载文件。

也可以用选项-o指定一个日志文件，这样日志信息就不会被打印到stdout了。

```
$ wget ftp://example_domain.com/somefile.img -O dloaded_file.img -o log
```

运行该命令，屏幕上不会出现任何内容。日志或进度信息被写入文件log，输出文件为dloaded_file.img。

由于不稳定的互联网连接，下载有可能被迫中断。我们可以将重试次数作为命令参数，这样一旦下载中断，wget在放弃之前还会继续进行多次尝试。

用-t指定重试次数：

```
$ wget -t 5 URL
```

或者要求wget不停地进行重试：

```
$ wget -t 0 URL
```

5.2.4　补充内容

wget还有一些其他选项，能够用以应对不同的情况。来看看其中的一部分。

1. 下载限速

当我们的下载带宽有限，却又有多个应用程序共享互联网连接时，进行大文件下载往往会榨干所有的带宽，严重阻滞其他进程。wget命令有一个内建的选项可以限定下载任务能够占有的最大带宽，从而保证其他应用程序流畅运行。

我们可以按照下面的方式使用--limit-rate对wget进行限速：

```
$ wget  --limit-rate 20k http://example.com/file.iso
```

在命令中用k（千字节）和m（兆字节）指定速度限制。

还可以指定最大下载配额（quota）。配额一旦用尽，下载随之停止。在下载多个文件时，对总下载量进行限制是有必要的，这能够避免在无意中占用过多磁盘空间。

使用--quota或-Q选项：

```
$ wget -Q 100m http://example.com/file1 http://example.com/file2
```

2. 断点续传

如果使用wget进行的下载在完成之前被中断，可以利用选项-c从断点开始继续下载：

```
$ wget -c URL
```

3. 复制整个网站（镜像）

wget有一个选项可以使其像爬虫一样以递归的方式遍历网页上所有的URL链接，并逐个下载。这样一来，我们就能够获得一个网站的所有页面。

要实现这个任务，可以按照下面的方式使用选项--mirror：

```
$ wget --mirror --convert-links exampledomain.com
```

或者

```
$ wget -r -N -l -k DEPTH URL
```

-l指定页面层级。这意味着wget只会向下遍历指定的页面级数。该选项要与-r（recursive，递归选项）一同使用。另外，-N表示使用文件的时间戳。URL表示欲下载的网站起始地址。-k或-convert-links指示wget将页面的链接地址转换为本地地址。

> 对网站进行镜像时，请三思而行。除非你被许可，否则只应出于个人使用的目的才可以这么做，而且不应该频繁地进行。

4. 访问需要认证的HTTP或FTP页面

一些网页需要HTTP或FTP认证，可以用--user和--password提供认证信息：

```
$ wget --user username --password pass URL
```

也可以不在命令行中指定密码，而由网页提示并手动输入密码，这就需要将--password改为--ask-password。

5.3 以纯文本形式下载网页

网页其实就是包含HTML标签和其他诸如Javascript、CSS等元素的HTML页面。HTML标记是网页的基础。我们也许需要解析网页来查找特定的内容，这时Bash就能派上用场了。当下载网页时，实际接收到的就是一个HTML文件。这种格式化的数据需要用浏览器查看。

在大多数情况下，解析文本文件要比解析HTML数据来得容易，因为无需再去剥离HTML标签。Lynx是一款颇有玩头的基于命令行的Web浏览器，可以利用它获取纯文本形式的网页。来看看这一切该如何实现。

实战演练

用lynx命令的-dump选项将网页内容以ASCII编码的形式存储到文本文件中：

```
$ lynx URL -dump > webpage_as_text.txt
```

这个命令会将页面中所有的超链接（<a href="link">）作为文本文件的页脚，单独放置在标题为References文本区域。这就省得我们再用正则表达式解析链接了。

例如：

```
$lynx -dump http://google.com > plain_text_page.txt
```

你可以用cat命令查看纯文本形式的网页：

```
$ cat plain_text_page.txt
```

5.4 cURL 入门

作为一款强力工具，cURL支持包括HTTP、HTTPS、FTP在内的众多协议。它还支持POST、

cookie、认证、从指定偏移处下载部分文件、参照页（referer）、用户代理字符串、扩展头部、限速、文件大小限制、进度条等特性。如果要和网页访问序列（web page usage sequence）以及数据检索自动化打交道，那么cURL定能助你一臂之力。这则攻略将为你展示cURL一系列最为重要的特性。

5.4.1　预备知识

在默认情况下，主流Linux发行版中并没有包含cURL，你得使用包管理器进行安装。不过多数发行版都默认附带了wget。

cURL通常将下载文件输出到stdout，将进度信息输出到stderr。要想避免显示进度信息，请使用--silent选项。

5.4.2　实战演练

curl命令的用途广泛，诸如下载、发送各种HTTP请求、指定HTTP头部都不在话下。来看看cURL如何做到这一切。

- 将下载的文件输出到终端（所下载的数据被写入stdout），使用下列命令：

```
$ curl URL
```

- 要避免curl命令显示进度信息，可以使用--silent选项：

```
$ curl URL --silent
```

- 选项 -O表明将下载数据写入文件，而非标准输出中。该文件采用的是从URL中解析出的文件名：

```
$ curl URL --silent -O
```

- 如果需要在下载过程中显示形如 # 的进度条，用 --progress代替 --silent。

```
$ curl http://slynux.org -o index.html --progress
################################## 100.0%
```

5.4.3　工作原理

cURL如果不将网页或文件写入stdout，就会将其写入和URL中所指定的文件名称相同的文件中。如果在URL中找不到文件名，则会产生错误。因此要确保URL指向的是远程文件。curl http://slynux.org -O -silent就会显示错误信息，这是因为无法从URL中解析出文件名。这种情况下，我们可以使用-o选项来手动指定文件名：

```
$ curl URL --silent -o new_filename
```

5.4.4　补充内容

在前面我们学习了如何下载文件以及将HTML页面打印到终端。cURL还包括一些高级选项，让我们接着进行研究。

1. 断点续传

cURL能够从特定的文件偏移处继续下载。可以通过指定一个偏移量来下载部分文件。

```
$ curl URL/file -C offset
```

偏移量是以字节为单位的整数。如果只是想断点续传，那么cURL不需要指定准确的字节偏移。要是你希望cURL推断出正确的续传位置，请使用选项 `-C -`，就像这样：

```
$ curl -C - URL
```

cURL会自动计算出应该从哪里开始续传。

2.用cURL设置参照页字符串

参照页（referer）[①]是位于HTTP头部中的一个字符串，用来标识用户是从哪个页面到达当前页面的。如果用户点击了网页A中的某个链接，那么用户就会转到网页B，网页B头部的参照页字符串会包含网页A的URL。

一些动态网页会在返回HTML页面前检测参照页字符串。例如，如果用户是通过Google搜索来到了当前网页，网页上会附带显示一个Google的logo；如果用户是通过手动输入URL来到当前网页，则显示另一个不同的页面。

网站的作者可以根据条件进行判断：如果参照页是www.google.com，那么就返回一个Google页面，否则返回其他页面。

可以用`curl`命令的 `--referer`选项指定参照页字符串：

```
$ curl --referer Referer_URL target_URL
```

例如：

```
$ curl --referer http://google.com http://slynux.org
```

3.用cURL设置cookie

我们可以用`curl`来指定并存储HTTP操作过程中使用到的cookie。

① referer其实应该是英文单词referrer，不过拼错的人太多了，所以编写标准的人也就将错就错了。

要指定cookie，使用 `--cookie "COOKIES"`选项。

cookies需要以`name=value`的形式来给出。多个cookie之间使用分号分隔。例如：

```
$ curl http://example.com --cookie "user=slynux;pass=hack"
```

如果要将cookie另存为一个文件，使用 `--cookie-jar`选项。例如：

```
$ curl URL --cookie-jar cookie_file
```

4.用cURL设置用户代理字符串

如果不指定用户代理（user agent），一些需要检验用户代理的网页就无法显示。你肯定碰到过有些陈旧的网站只能在Internet Explorer（IE）下正常工作。如果使用其他浏览器，这些网站就会提示说它只能用IE访问。这是因为这些网站检查了用户代理。你可以用`curl`来设置用户代理。

cURL的 `--user-agent`或 `-A`选项用于设置用户代理：

```
$ curl URL --user-agent "Mozilla/5.0"
```

其他HTTP头部信息也可以通过cURL来发送。用`-H"头部信息"`传递多个头部信息。例如：

```
$ curl -H "Host: www.slynux.org" -H "Accept-language: en" URL
```

> 各类网页浏览器和爬虫使用了很多不同的用户代理字符串。你可以在这里找到其中的一部分：http://www.useragentstring.com/pages/useragentstring.php。

5. 限定cURL可占用的带宽

如果带宽有限，又有多个用户共享，为了平稳流畅地分享带宽，我们可以用`--limit-rate`限制cURL的下载速度：

```
$ curl URL --limit-rate 20k
```

在命令中用`k`（千字节）和`m`（兆字节）指定下载速度限制。

6. 指定最大下载量

可以用`--max-filesize`选项指定可下载的最大文件大小：

```
$ curl URL --max-filesize bytes
```

如果文件大小超出限制，命令返回一个非0的退出码。如果命令正常运行，返回0。

7. 用cURL进行认证

可以用cURL的选项 -u完成HTTP或FTP认证。

-u username:password可用来指定用户名和密码。它也可以不指定密码，而在后续的执行过程中按照提示输入密码。例如：

```
$ curl -u user:pass http://test_auth.com
```

如果你喜欢经提示后输入密码，只需要使用 -u username即可。例如：

```
$ curl -u user http://test_auth.com
```

8. 只打印响应头部信息（不包括数据部分）

只打印响应头部（response header）有助于进行各种检查或统计。例如，如果要检查某个页面是否能够打开，并不需要下载整个页面内容。只用读取HTTP响应头部就能够知道这个页面是否可用。

检查HTTP头部的一个用法就是在下载之前先获知文件大小。我们可以在下载之前，通过检查HTTP头部中的 Content-Length 参数来得知文件的长度。同样还可以从头部检索出其他一些有用的参数。Last-Modified参数能告诉我们远程文件最后的改动时间。

通过 -I或--head 就可以只打印HTTP头部信息，而无须下载远程文件。例如：

```
$ curl -I http://slynux.org
HTTP/1.1 200 OK
Date: Sun, 01 Aug 2010 05:08:09 GMT
Server: Apache/1.3.42 (Unix) mod_gzip/1.3.26.1a mod_log_bytes/1.2 mod_bwlimited/1.4
mod_auth_passthrough/1.8 FrontPage/5.0.2.2635 mod_ssl/2.8.31 OpenSSL/0.9.7a
Last-Modified: Thu, 19 Jul 2007 09:00:58 GMT
ETag: "17787f3-3bb0-469f284a"
Accept-Ranges: bytes
Content-Length: 15280
Connection: close
Content-Type: text/html
```

5.4.5 参考

参考5.13节。

5.5 从命令行访问 Gmail

Gmail（http://mail.google.com）是来自Google的一项被广泛使用的免费电子邮件服务。你可

5

以使用经过认证的RSS feed来读取个人邮件。我们也能够使用发件人姓名以及邮件主题来解析RSS feed。这样便无须打开网页浏览器就能够查看收件箱中的未读邮件。

5.5.1　实战演练

来看下面这个脚本文件，它的作用是通过解析Gmail的RSS feed来显示未读的邮件：

```
#!/bin/bash
#用途：获取Gmail工具

username='PUT_USERNAME_HERE'
password='PUT_PASSWORD_HERE'

SHOW_COUNT=5 # 需要显示的未读邮件数量

echo
curl  -u $username:$password --silent "https://mail.google.com/mail/feed/atom" | \
tr -d '\n' | sed 's:</entry>:\n:g' |\
 sed -n 's/.*<title>\(.*\)<\/title.*<author><name>\([^<]*\)<\/name><email>
\([^<]*\).*/From: \2 [\3] \nSubject: \1\n/p' | \
head -n $(( $SHOW_COUNT * 3 ))
```

输出如下:

```
$ ./fetch_gmail.sh
From: SLYNUX [ slynux@slynux.com ]
Subject: Book release - 2

From: SLYNUX [ slynux@slynux.com ]
Subject: Book release - 1
.
……共5封未读邮件
```

> 如果你的Gmail使用双重身份认证，那就必须为此脚本生成一个新建，并使用。你的普通密码不会起作用。

5.5.2　工作原理

这个脚本通过用户认证来使用cURL来下载RSS feed。用户认证信息由`-u username:password`参数提供。也可以只使用`-u user`，这样就不用事先提供密码，而是由cURL在运行时以交互的方式索要密码。我们将管道命令拆分成不同的部分来解释这个脚本的工作原理。

`tr -d '\n'`移除了所有的换行符，这样我们就可以将`\n`作为定界符来重建邮件项。

sed 's:</entry>:\n:g'将每一处</entry>替换成换行符，以保证每一条邮件项独立成行，以便逐条解析邮件。通过查看https://mail.google.com/mail/feed/atom页面源代码中用于RSS feed的XML标记，我们可以看出每一条邮件项都对应于<entry> TAGS </entry>。

该脚本接下来的部分如下：

```
sed 's/.*<title>\(.*\)<\/title.*<author><name>\([^<]*\)<\/name><email>
\([^<]*\).*/Author: \2 [\3] \nSubject: \1\n/'
```

脚本用<title>\(.*\)<\/title匹配邮件标题，用<author><name>\([^<]*\)<\/ name>匹配发件人姓名，<email>\([^<]*\)匹配发件人电子邮件地址。然后使用反向引用：

```
Author: \2 [\3] \nSubject: \1\n
```

用来将匹配内容以易于阅读的格式来替换原先的邮件项。\1对应于第一处匹配，\2对应于第二处匹配，依次类推。

SHOW_COUNT=5用来设置需要在终端中显示的未读邮件数量。

head用来显示SHOW_COUNT*3[①]行文本。SHOW_COUNT乘以3是因为每一封未读邮件的相关信息需要占用3行。

5.5.3　参考

- 5.4节讲解了curl命令。
- 4.5节讲解了sed命令。

5.6　解析网站数据

消除不必要的繁枝末节有助于解析网页数据。sed和awk是完成这项工作的主要工具。在第4章有关grep的攻略中，我们见到过一份演员评级列表。那个列表就是通过解析http://www.johntorres.net/BoxOfficefemaleList.html得到的。

让我们看看如何使用文本处理工具来解析同样的数据。

5.6.1　实战演练

请看下面用于从网站解析演员详细信息的命令：

```
$ lynx -dump -nolist http://www.johntorres.net/BoxOfficefemaleList.html  | \
grep -o "Rank-.*" | \
sed -e 's/ *Rank-\([0-9]*\) *\(.*\)/\1\t\2/' | \
```

① 这里的SHOW_COUNT*3行文本并不包括脚本开始部分由echo生成的那一行（空行）。

```
sort -nk 1 > actresslist.txt
```

输入如下：

```
# 由于版面空间限制，只显示前3位演员的信息
1   Keira Knightley
2   Natalie Portman
3   Monica Bellucci
```

5.6.2　工作原理

Lynx是一个基于命令行的网页浏览器。它并不会输出一堆原始的HTML代码，而是能够显示网站的文本版本，这个文本版和我们在浏览器中看到的页面一模一样。这样一来，就免去了移除HTML标签的工作。这里用到了lynx的-nolist选项，这是因为不需要给每个链接自动加上数字标号。按照下面的方法用sed解析并格式化以Rank开头的行：

```
sed -e 's/ *Rank-\([0-9]*\) *\(.*\)/\1\t\2/'
```

然后再根据评级情况对这些行进行排序。

5.6.3　参考

- 4.5节讲解了sed命令。
- 5.3节讲解了lynx命令。

5.7　图片抓取器及下载工具

当需要下载某个网页上所有的图片时，**图片抓取器**（image crawler）能够帮上我们的大忙。用不着把HTML源码翻个底朝天来摘出所有的图片，我们可以用脚本解析图像文件并将它们自动下载下来。来看看这是如何实现的。

5.7.1　实战演练

用于从网页上抓取并下载图片的Bash脚本如下：

```
#!/bin/bash
#用途:图片下载工具
#文件名: img_downloader.sh

if [ $# -ne 3 ];
then
  echo "Usage: $0 URL -d DIRECTORY"
  exit -1
fi
```

```
for i in {1..4}
do
  case $1 in
  -d) shift; directory=$1; shift ;;
   *) url=${url:-$1}; shift;;
  esac
done

mkdir -p $directory;
baseurl=$(echo $url | egrep -o "https?://[a-z.]+")

echo Downloading $url
curl -s $url | egrep -o "<img src=[^>]*>" |
sed 's/<img src=\"\([^"]*\).*/\1/g' > /tmp/$$.list

sed -i "s|^/|$baseurl/|" /tmp/$$.list

cd $directory;

while read filename;
do
  echo Downloading $filename
  curl -s -O "$filename" --silent

done < /tmp/$$.list
```

使用方法：

```
$ ./img_downloader.sh http://www.flickr.com/search/?q=linux -d images
```

5.7.2 工作原理

上述图片下载器脚本首先解析HTML页面，除去 `<img>` 之外的所有标签，然后从 `<img>` 标签中解析出`src="URL"` 并将图片下载到指定的目录中。这个脚本接受一个网页URL和用于存放图片的目录路径作为命令行参数。`[ $# -ne 3 ]` 用于检查脚本参数数量是否为3个。如果不是，它就会退出运行并返回脚本用法说明。

如果参数是3个，那么就解析URL和目标目录。实现方法如下：

```
while [ -n "$1" ]
do
  case $1 in
  -d) shift; directory=$1; shift ;;
   *) url=${url:-$1}; shift;;
esac
done
```

这里用到了while循环。有几个参数，它就循环几次。case语句会对第一个参数（$1）求值，以便匹配-d或是其他需要进行检查的参数。采用这种方法来解析命令行参数的好处在于，可以将-d置于命令行中的任意位置：

```
$ ./img_downloader.sh -d DIR URL
```

或者

```
$ ./img_downloader.sh URL -d DIR
```

shift用来向左移动参数。当使用shift后，$2的值就被赋给 $1；如果再次使用shift，则 $3的值被赋给 $1，往后依次类推，因此通过 $1就可以对所有的参数进行求值。

如果匹配-d，显然接下来的参数就是目标目录。*)对应默认匹配（default match）。它能够匹配除了-d之外的任何内容。因此在默认匹配中，如果$1="" 或 $1=URL，就需要采用$1=URL，避免""将变量url覆盖掉，因此我们使用了url=${url:-$1}。如果url不为空，它会返回URL值，否则返回$1的值。

egrep -o "<img src=[^>]*>"只打印包括属性值在内的 <img> 标签。[^>]*用来匹配除 > 之外的所有字符，结果就是 <img src="image.jpg" .... >。

sed 's/<img src=\"\([^"]*\).*/\1/g'对 src="url"进行解析，这样就可以从已经解析出来的 <img> 标签中得到所有图像文件的URL。

图像文件源路径有两种类型：相对路径和绝对路径。**绝对路径**包含以http:// 或 https:// 起始的完整URL，**相对路径**则以 / 或图像文件名起始。绝对路径例如：http://example.com/image.jpg；相对路径例如：/image.jpg。

对于以/起始的相对路径，应该用基址URL（base URL）把它转换为 http://example.com/image.jpg。我们一开始通过解析将基址URL存入变量baseurl中，然后用 "s|^/|$baseurl/|" /tmp/$$.list将所有的 / 替换成baseurl。

```
"s|^/|$baseurl/|" /tmp/$$.list
```

while循环用来对图片的URL列表进行逐行迭代，并用curl下载图像文件。与curl一同使用的 --silent 选项可避免下载进度信息出现在屏幕上。

5.7.3　参考

- 5.4节讲解了curl命令。
- 4.5节讲解了sed命令。
- 4.3节讲解了grep命令。

5.8 网页相册生成器

Web开发人员通常都会为网站设计相册页面，这些页面上包含着多个图像缩略图。点击缩略图，就会出现一幅放大的图片。但如果需要很多图片，每一次都得复制 `<img>` 标签、调整图片大小来创建缩略图、把调整好的图片放进缩略图目录，诸如此类的活儿实在烦琐。我们可以写一个简单的Bash脚本将这些重复的工作以自动化的方式来处理。这样一来，创建缩略图、将缩略图放入对应的目录、生成对应于`<img>`标签的代码片段都可以在短短几秒钟内自动搞定。这则攻略正是要告诉你如何实现刚才所说的一切。

5.8.1 预备知识

要完成这项任务，可以通过`for`循环对当前目录下的所有图片进行迭代。这需要借助一些常见的Bash工具，如`cat`和`convert`（来自`Image Magick`软件包）。我们将用所有的图片生成一个HTML相册index.html（见图5-1）。



图 5-1

5.8.2 实战演练

生成HTML相册页面的Bash脚本如下：

```
#!/bin/bash
#文件名： generate_album.sh
#用途： 用当前目录下的图片创建相册
```

```
echo "Creating album.."
mkdir -p thumbs
cat <<EOF1 > index.html
<html>
<head>
<style>

body
{
  width:470px;
  margin:auto;
  border: 1px dashed grey;
  padding:10px;
}

img
{
  margin:5px;
  border: 1px solid black;

}
</style>
</head>
<body>
<center><h1> #Album title </h1></center>
<p>
EOF1

for img in *.jpg;
do
  convert "$img" -resize "100x" "thumbs/$img"
  echo "<a href=\"$img\" ><img src=\"thumbs/$img\" title=\"$img\" /></a>" >>
index.html
done

cat <<EOF2 >> index.html

</p>
</body>
</html>
EOF2

echo Album generated to index.html
```

运行脚本：

```
$ ./generate_album.sh
Creating album..
Album generated to index.html
```

5.8.3 工作原理

脚本的起始部分用于生成HTML页面的头部。

接下来，脚本将一直到`EOF1`的这部分内容（不包括`EOF1`）重定向到index.html：

```
cat <<EOF1 > index.html
contents...
EOF1
```

页面头部包括HTML和CSS样式表单。

`for img in *.jpg;`对每一个文件进行迭代，并执行相应的操作。

`convert "$img" -resize "100x" "thumbs/$img"`将创建宽度为100像素的图像缩略图。

下面的语句会生成所需的`<img>`标记并将其追加到index.html中：

```
echo "<a href=\"$img\" ><img src=\"thumbs/$img\" title=\"$img\" /></a>" >> index.html
```

最后再用`cat`添加HTML页脚。

5.8.4 参考

1.6节讲解了EOF和`stdin`重定向。

5.9 Twitter 命令行客户端

Twitter不仅是最流行的微博平台，同时也是最时髦的在线社交媒体。我们可以使用Twitter API从命令行中读取自己发布的微博。来看看如何实现。

5.9.1 预备知识

最近Twitter已经不再允许用户使用普通的HTTP认证（plain HTTP Authentication）进行登录了，我们必须使用OAuth进行自认证（authenticate ourselves）。完整地讲解OAuth超出了本书的范围，因此我们会利用一个代码库以便在Bash脚本中可以轻松使用OAuth。

(1) 从https://github.com/livibetter/bash-oauth/archive/master.zip处下载bash-oauth库，将其解压缩到任意目录中。

(2) 进入该目录中的bash-oauth-master子目录，以root身份执行make install-all。

(3) 进入https://dev.twitter.com/apps/new注册新的应用，以便能够使用OAuth，如图5-2所示。

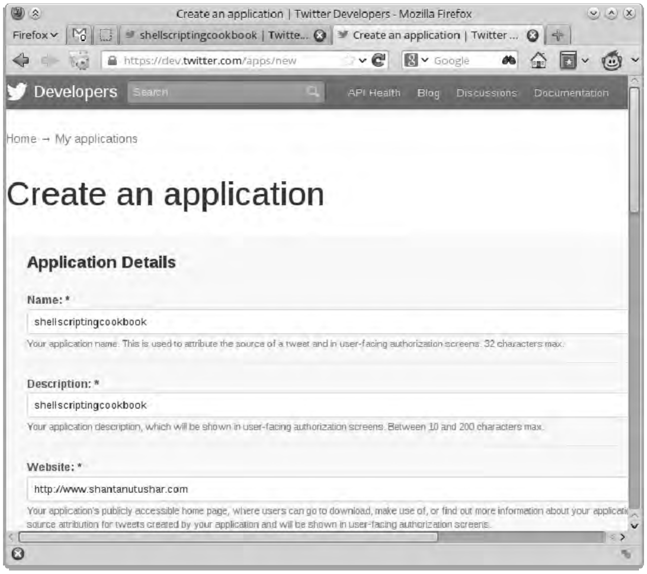


图　5-2

(4) 注册完新的应用之后，进入应用设置，将**Access type**更改为**Read and Write**，如图5-3所示。

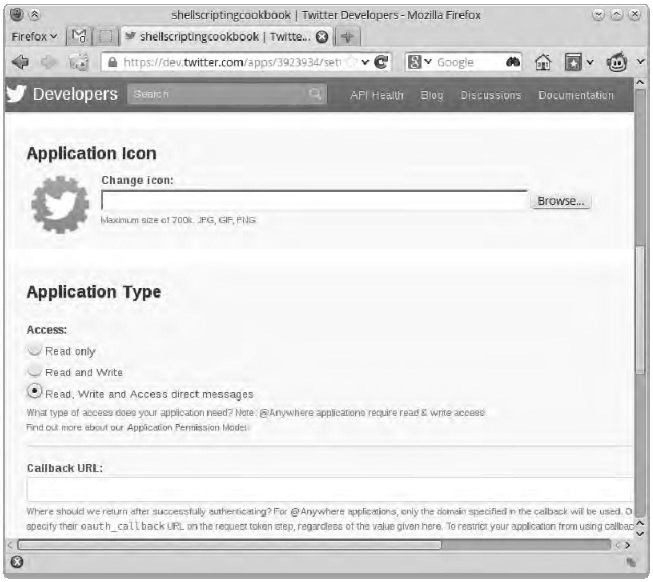


图　5-3

(5) 进入应用的Details部分，注意两个地方：Consumer Key和Consumer Secret，以便在脚本中对其进行替换，如图5-4所示。

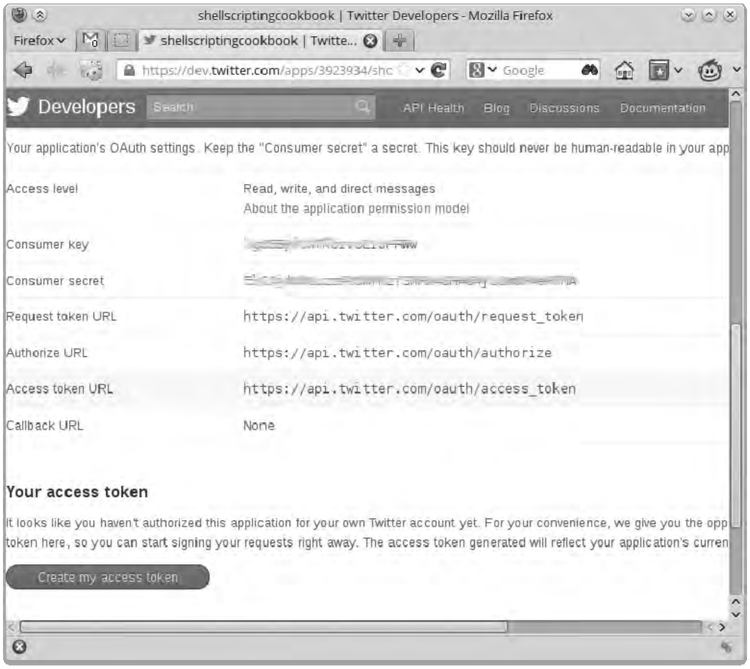


图 5-4

好，接下来就该编写脚本了。

5.9.2 实战演练

接下来编写使用代码库的Bash脚本：

```
#!/bin/bash
#文件名： twitter.sh
#用途:twitter客户端基本版

oauth_consumer_key=YOUR_CONSUMER_KEY
oauth_consumer_secret=YOUR_CONSUMER_SECRET
config_file=~/.$oauth_consumer_key-$oauth_consumer_secret-rc

if [[ "$1" != "read" ]] && [[ "$1" != "tweet" ]];
then
```

```
  echo -e "Usage: $0 tweet status_message\n   OR\n      $0 read\n"
  exit -1;
fi

source TwitterOAuth.sh
TO_init

if [ ! -e $config_file ]; then
 TO_access_token_helper
 if (( $? == 0 )); then
   echo oauth_token=${TO_ret[0]} > $config_file
   echo oauth_token_secret=${TO_ret[1]} >> $config_file
 fi
fi

source $config_file

if [[ "$1" = "read" ]];
then
  TO_statuses_home_timeline '' 'shantanutushar' '10'
  echo $TO_ret | sed 's/<\([a-z]\)/\n<\1/g' | \
grep -e '^<text>' -e '^<name>' | sed 's/<name>/\ - by /g' | \
sed 's$</*[a-z]*>$$g'

elif [[ "$1" = "tweet" ]];
then
  shift
  TO_statuses_update '' "$@"
  echo 'Tweeted :)'
fi
```

运行脚本：

```
$./twitter.sh read
Please go to the following link to get the PIN:
https://api.twitter.com/oauth/authorize?oauth_token=GaZcfsdnhMO4HiBQuUTdeLJAzeaUam
nOljWGnU
PIN: 4727143
Now you can create, edit and present Slides offline.
 - by A Googler
$./twitter.sh tweet "I am reading Packt Shell Scripting Cookbook"
Tweeted :)
$./twitter.sh read | head -2
I am reading Packt Shell Scripting Cookbook
 - by Shantanu Tushar Jha
```

5.9.3 工作原理

首先，使用source命令将TwitterOAuth.sh库包含在脚本中，这样就可以利用其中定义好的函数访问Twitter了。我们需要调用库函数TO_init，以便库完成自身的初始化操作。

所有的应用在用户首次使用的时候都需要获取一个OAuth令牌（token）以及令牌密钥（token secret）。我们需要检查是否已经获取。如果没有，则调用库函数TO_access_token_helper。得到之后，将其保存在config文件中，以后再执行脚本时，只需对该文件执行source命令就可以了。

我们使用库函数TO_statuses_home_timeline获取Twitter中发布的内容，该函数会读取Twitter，将得到的XML文件内容存入变量TO_ret，然后使用sed为每一个标签（tag）加上一个换行符，过滤出<text>和<name>标签，最后再将这些标签删除，替换成便于用户阅读的文本。

要发布新的微博，可以使用库函数TO_statuses_update。如果该函数的第一个参数为空，则表明使用默认格式，要发布的内容可以作为第二个参数传递给函数。

5.9.4 参考

- 5.4节讲解了sed命令。
- 4.3节讲解了grep命令。

5.10 基于 Web 后端的定义查询工具

网上有大量提供了API的词典工具，利用这些API可以获得能被机器读取（machine-readable）的词汇定义。让我们来使用其中的一种API来编写一个词汇定义查询脚本。

5.10.1 预备知识

我们用curl、sed和grep来编写词汇查询工具。有关词典的网站数不胜数，你可以注册并免费使用它们的API（限于个人用途）。在这里，我们使用Merriam-Webster的词典API。请按照下列步骤执行。

(1) 进入http://www.dictionaryapi.com/register/index.htm注册账户。选择Collegiate Dictionary和Learner’s Dictionary，如图5-5所示。



图　5-5

(2) 使用新创建的用户登录，进入My Keys获取密钥。记下Learner’s Dictionary的密钥，如图5-6所示。



图　5-6

5.10.2 实战演练

来看下面这段定义工具脚本：

```
#!/bin/bash
#文件名: define.sh
#用途:用于从 dictionaryapi.com 获取词汇定义

apikey=YOUR_API_KEY_HERE

if  [ $# -ne 2 ];
then
  echo -e "Usage: $0 WORD NUMBER"
  exit -1;
fi

curl --silent
http://www.dictionaryapi.com/api/v1/references/learners/xml/$1?key=$apikey | \
grep -o \<dt\>.*\</dt\> | \
sed 's$</*[a-z]*>$$g' | \
head -n $2 | nl
```

运行脚本：

```
$ ./define.sh usb 1
     1  :a system for connecting a computer to another device (such as a printer,
keyboard, or mouse) by using a special kind of cord a USB cable/port USB is an
abbreviation of "Universal Serial Bus."How it works...
```

5.10.3 工作原理

我们通过指定API key（$apikey）来使用curl从词典API页面获取数据以及需要查找定义的词汇（$1）。包含定义的查询结果位于<tag>标签中，可以使用grep来将其选中，然后使用sed删除标签。接着从定义中提取所需要的行数，利用nl在行前加上行号。

5.10.4 参考

- 4.5节讲解了sed命令。
- 4.3节讲解了grep命令。

5.11 查找网站中的无效链接

一些人采用人工方式来检查网站上的每一个页面，以便找出无效的链接。这对于页面不多的

小网站来说还算可行。但是随着页面数量增多，这种方式就不现实了。要是能够将查找无效链接的工作自动化，那可就轻松太多了。我们可以利用HTTP处理工具来查找无效的链接。来看看这是怎样实现的。

5.11.1　预备知识

要识别链接并从中找出无效链接，我们可以使用`lynx`和`curl`。`lynx`有一个选项`-traversal`，能够以递归方式访问网站页面并建立网站中所有超链接的列表。我们可以用`curl`验证每一个链接的有效性。

5.11.2　实战演练

利用`curl`查找网页上无效链接的Bash脚本如下：

```
#!/bin/bash
#文件名：find_broken.sh
#用途：查找网站中的无效链接

if [ $# -ne 1 ];
then
  echo -e "$Usage: $0 URL\n"
  exit 1;
fi


echo Broken links:

mkdir /tmp/$$.lynx
cd /tmp/$$.lynx

lynx -traversal $1 > /dev/null
count=0;

sort -u reject.dat > links.txt

while read link;
do
  output=`curl -I $link -s | grep "HTTP/.*OK"`;
  if [[ -z $output ]];
  then
    echo $link;
    let count++
  fi
done < links.txt

[ $count -eq 0 ] && echo No broken links found.
```

5.11.3　工作原理

`lynx -traversal URL`会在工作目录下生成数个文件，其中包括`reject.dat`，该文件包含网站中的所有链接。`sort -u`用来建立一个不包含重复项的列表。然后我们迭代每一个链接，并通过`curl -I`检验接收到的响应头部。如果响应头部的第一行包含`HTTP/1.0 200 OK`，就表示该链接正常。显示其他响应信息则表示该链接失效，并将其输出到屏幕。

> 从名称上来看，reject.dat中包含的应该是无效URL的列表。但其实并非如此，lynx是将所有的URL全都放到了这个文件中。
>
> lynx还生成了一个名为traverse.errors的文件，其中包含了所有在浏览过程中存在问题的URL。但是lynx只会将返回HTTP 404(not found)的URL，因此会遗漏那些存在其他类型错误的URL（例如HTTP 403 Forbidden）。这就是为什么要手动检查返回状态的原因。

5.11.4　参考

- 5.3节讲解了`lynx`命令。
- 5.4节讲解了`curl`命令。

5.12　跟踪网站变动

对于Web开发人员和用户来说，跟踪网站的变动情况不无益处。但定期手动检查既麻烦也不现实，因此，我们可以编写一个定期运行的变动跟踪器（change tracker）。一旦发生变动，跟踪器便会发出声音或发送提示信息。下面看一看如何编写基础的网站变动跟踪器。

5.12.1　预备知识

用Bash脚本跟踪网站变动意味着要在不同的时间检索网站，然后用`diff`命令进行比对。我们可以使用`curl`和`diff`来实现。

5.12.2　实战演练

结合各种命令来跟踪网页变更的Bash脚本如下：

```
#!/bin/bash
#文件名：change_track.sh
#用途：跟踪网页更新

if [ $# -ne 1 ];
then
  echo -e "$Usage: $0 URL\n"
  exit 1;
fi

first_time=0
#非首次运行

if [ ! -e "last.html" ];
then
  first_time=1
  #首次运行
fi

curl --silent $1 -o recent.html

if [ $first_time -ne 1 ];
then
  changes=$(diff -u last.html recent.html)
  if [ -n "$changes" ];
  then
    echo -e "Changes:\n"
    echo "$changes"
  else
    echo -e "\nWebsite has no changes"
  fi
else
  echo "[First run] Archiving.."

fi


cp recent.html last.html
```

分别观察一下网页发生变动与没有变动时脚本track_changes.sh的输出。

- 第一次运行：

```
$ ./track_changes.sh http://web.sarathlakshman.info/test.html
[First run] Archiving..
```

- 第二次运行：

```
$ ./track_changes.sh http://web.sarathlakshman.info/test.html
Website has no changes
```

- 在网页变动后，第三次运行：

```
$ ./test.sh http://web.sarathlakshman.info/test_change/test.html
Changes:

--- last.html 2010-08-01 07:29:15.000000000 +0200
+++ recent.html       2010-08-01 07:29:43.000000000 +0200
@@ -1,3 +1,4 @@
<html>
+added line :)
<p>data</p>
</html>
```

5.12.3　工作原理

脚本用`[ ! -e "last.html" ];`检查自己是否是首次运行。如果`last.html`不存在，那就意味着这是首次运行，因此要下载网页并将其复制为`last.html`。

如果不是第一次运行，那么脚本应该下载一个新的网页副本（`recent.html`），然后用`diff`检查差异。如果有变化，则打印出变更信息并将`recent.html`复制成`last.html`。

5.12.4　参考

5.4节讲解了`curl`命令。

5.13　以 POST 方式发送网页并读取响应

POST和GET是HTTP协议中用于发送或检索信息的两种请求类型。在GET请求方式中，我们利用网页的URL来发送参数（名称-值）。而在POST请求方式中，就不是这样了。POST方式用于提交表单。例如提交用户名、密码以及检索登录页面等。

当编写基于网页检索的脚本时，用POST方式发送页面很常见。让我们来看一看POST的使用方法。在编写解析网站数据的shell脚本的过程中，通过POST方式发送数据并检索输出结果来实现HTTP GET和POST请求自动化，是一项非常重要的任务。

5.13.1　预备知识

这里我们使用了一个测试网站（http://book.sarathlakshman.com/lsc/mlogs/）来提交当前的用户信息，如主机名和用户名。在网站主页上有两个文本域：HOSTNAME和USER以及一个SUBMIT

按钮。当用户输入主机名和用户名，点击SUBMIT按钮后，信息会被存储在网站中。这个过程可以使用一行curl（或wget）自动化POST请求来实现。来看看具体的做法。

5.13.2　实战演练

使用curl发送POST请求并读取网站的响应（HTML格式）：

```
$ curl URL -d "postvar=postdata2&postvar2=postdata2"
```

例如：

```
$ curl http://book.sarathlakshman.com/lsc/mlogs/submit.php -d
"host=test-host&user=slynux"
<html>
You have entered :
<p>HOST : test-host</p>
<p>USER : slynux</p>
<html>
```

curl会打印出响应页面。

-d表示以POST方式提交用户数据。-d的字符串参数的形式类似于GET请求。每对var=value之间用&分隔。

也可以利用wget，使用它的–post-data "string"来提交数据。例如：

```
$ get http://book.sarathlakshman.com/lsc/mlogs/submit.php --post-data
"host=test-host&user=slynux" -O output.html
$ cat output.html
<html>
You have entered :
<p>HOST : test-host</p>
<p>USER : slynux</p>
<html>
```

"名称–值"组合的格式同cURL中的一样。

> -d参数的内容应该以引用的形式给出。否则，& 会被shell解读为该命令需要作为后台进程运行。

5.13.3　工作原理

如果你查看网站的源代码（使用网页浏览器的**查看源代码**选项），你会发现一个与下面类似的HTML表单：

```
<form action="http://book.sarathlakshman.com/lsc/mlogs/submit.php" method="post" >

<input type="text" name="host" value="HOSTNAME" >
<input type="text" name="user" value="USER" >
<input type="submit" >
</form>
```

其中 `http://book.sarathlakshman.com/lsc/mlogs/submit.php`是目标URL。当用户输入信息并点击SUBMIT按钮时，主机名和用户名就以POST请求的方式被发送到submit.php中，然后对应的响应页面被返回到浏览器。

5.13.4 参考

- 5.4节讲解了`curl`命令。
- 5.2节讲解了`wget`命令。

第 6 章 B计划

6

本章内容

- 用tar归档
- 用cpio归档
- 用gzip压缩数据
- 用zip归档和压缩
- 更快速的归档工具pbzip2
- 创建压缩文件系统
- 用raync备份系统快照
- 用Git进行基于版本控制的备份
- 用fsarchiver创建全盘镜像

6.1 简介

提取快照和备份数据都是我们的日常工作，就服务器或大型数据存储系统而言，定期备份更是不可小视。shell脚本是实现备份自动化最简单的方法之一，如果不能实现自动化，备份的用处就大打折扣了。采用各种压缩方式也值得一试，这样我们便能够减少备份文件的体积。加密是另一种保护数据的常用方法。为了减少加密数据的大小，文件在加密之前通常都要先进行归档和压缩。有很多标准加密算法可以使用，而且也都有相应的shell工具。本章包含了这方面的各类攻略，包括创建和维护文件或文件夹归档、压缩格式以及加密技术。接下来让我们看看这些内容。

6.2 用 tar 归档

tar命令可以对文件进行归档。它最初是设计用来将数据存储在磁带上。tar可以将多个文件和文件夹保存为单个文件，同时还能保留所有的文件属性，如所有者、权限等。由tar创建的文件通常称为Tarball。在这则攻略里，我们将学习如何使用tar归档。

6.2.1 预备知识

所有Unix类操作系统中都默认包含tar命令。它语法简单，文件格式具备可移植性。tar支

持的参数包括：A、c、d、r、u、x、f和v。其中每一个参数都可以依据不同的用途单独使用。

6.2.2　实战演练

我们可以使用tar创建归档文件，对已有的归档文件进行操作。

(1) 用tar对文件进行归档：

```
$ tar -cf output.tar [SOURCES]
```

例如：

```
$ tar -cf output.tar file1 file2 file3 folder1 ..
```

(2) 使用选项-t列出归档文件中所包含的文件：

```
$ tar -tf archive.tar
file1
file2
```

(3) 如果需要在归档或列出归档文件列表时获知更多的细节信息，可以使用-v或-vv参数。这个特性叫做“冗长模式”（verbose），对于大多数命令而言，该模式会在终端中输出更多的细节。例如，利用冗长模式，你可以得到诸如文件权限、所有者所属的分组、文件修改日期等信息。

```
$ tar -tvf archive.tar
-rw-rw-r-- shaan/shaan       0 2013-04-08 21:34 file1
-rw-rw-r-- shaan/shaan       0 2013-04-08 21:34 file2
```


> 文件名必须紧跟在-f之后，而且-f应该是选项中的最后一个。假如你希望使用冗长模式，那就应该像这样使用：
>
> ```
> $ tar -cvf output.tar file1 file2 file3 folder1 ..
> ```

6.2.3　工作原理

命令中的 -c代表“创建文件”（creat file），-f代表“指定文件名”（specify filename）。

我们可以在SOURCE处指定目录或文件名。也可以使用文件名列表或者诸如 *.txt这类通配符来指定。命令执行完毕后，tar会将源文件归档为output.tar。

不能传递上百个文件或文件夹作为 tar 的参数，毕竟参数数量不是无限制的。如果有很多文件需要归档，那么使用追加选项（详见下文）要更安全些。

6.2.4 补充知识

让我们再来看看tar命令的其他特性。

1. 向归档文件中添加文件

有时候，我们可能需要向已存在的归档文件再添加一些文件，这时可以使用追加选项-r。

要向已有的归档文件中添加一个文件：

```
$ tar -rvf original.tar new_file
```

创建一个其中包含有文本文件的归档：

```
$ tar -cf archive.tar hello.txt
```

用下面的方法列出归档文件中的内容：

```
$ tar -tf archive.tar
hello.txt
```

接着向归档文件中再添加另一个文件，并列出归档内容：

```
$ tar -rf archive.tar world.txt
$ tar -tf archive.tar
hello.txt
world.txt
```

这个归档文件中现在包含了两个文件。

2. 从归档文件中提取文件或文件夹

下面的命令将归档文件的内容提取到当前目录中：

```
$ tar -xf archive.tar
```

选项-x表示提取（exact）。

使用-x时，tar命令将归档文件中的内容提取到当前目录。我们也可以用选项-C来指定需要将文件提取到哪个目录：

```
$ tar -xf archive.tar -C /path/to/extraction_directory
```

这个命令将归档文件的内容提取到指定目录中。它提取的是归档文件中的全部内容。我们可以通过将文件名指定为命令行参数来提取特定的文件：

```
$ tar -xvf file.tar file1 file4
```

上面的命令只提取file1和file4，忽略其他文件。

3. 在tar中使用stdin和stdout

进行归档时，我们可以将stdout指定为输出文件，这样另一个命令就可以通过管道来读取（作为stdin），然后进行其他处理或提取内容。

当通过安全shell（Secure Shell，SSH）连接传输数据时，这招很管用。例如：

```
$ tar cvf - files/ | ssh user@example.com "tar xv -C Documents/"
```

在上面的例子中，对files目录中的内容进行了归档并输出到stdout（由'-'指明）。

4. 拼接两个归档文件

我们可以用 -A选项轻松地合并多个tar文件。

假设我们现在有两个tar文件：file1.tar和file2.tar。可以按照下面的方法将file2.tar的内容合并到file1.tar中：

```
$ tar -Af file1.tar file2.tar
```

查看内容，验证操作是否成功：

```
$ tar -tvf file1.tar
```

5. 通过检查时间戳来更新归档文件中的内容

添加选项可以将指定的任意文件加入到归档文件中。如果同名文件已经存在，那么结果就是在归档文件中包含了两个同名的文件。我们可以用更新选项-u指明：只有比归档文件中的同名文件更新时才会被添加。

```
$ tar -tf archive.tar
filea
fileb
filec
```

上面的命令将列出归档文件中的内容。

仅当filea自上次被加入archive.tar后出现了变动才对其进行追加，可以使用：

```
$ tar -uf archive.tar filea
```

如果两个filea的时间戳相同，则什么都不会发生。

可用touch命令修改文件的时间戳，然后再用tar命令：

```
$ tar -uvvf archive.tar filea
-rw-r--r-- slynux/slynux     0 2010-08-14 17:53 filea
```

因为时间戳比归档文件中的同名文件更新，因此执行追加操作。验证如下：

```
$ tar -tf archive.tar
-rw-r--r-- slynux/slynux     0 2010-08-14 17:52 filea
-rw-r--r-- slynux/slynux     0 2010-08-14 17:52 fileb
-rw-r--r-- slynux/slynux     0 2010-08-14 17:52 filec
-rw-r--r-- slynux/slynux     0 2010-08-14 17:53 filea
```

如你所见，一个新的filea被加入了归档文件中。当从中提取文件时，tar会挑选最新的filea进行提取。

6. 比较归档文件与文件系统中的内容

有时候需要知道归档文件中的文件与文件系统中的同名文件是否相同。选项-d可以打印出两者之间的差别：

```
$ tar -df archive.tar
afile: Mod time differs
afile: Size differs
```

7. 从归档文件中删除文件

我们可以用 --delete选项从给定的归档文件中删除文件。例如：

```
$ tar -f archive.tar --delete file1 file2 ..
```

或者

```
$ tar --delete --file archive.tar [FILE LIST]
```

来看另外一个例子：

```
$ tar -tf archive.tar
filea
fileb
filec
```

删除filea：

```
$ tar --delete --file archive.tar filea
$ tar -tf archive.tar
fileb
filec
```

8. 压缩tar归档文件

tar命令只能用来对文件进行归档，它并不具备压缩功能。出于这个原因，多数用户在使用归档文件时都会对文件采用某种形式的压缩，这样就能够显著减少文件的体积。归档文件通常被压缩成下列格式之一：

- ❑ `file.tar.gz`
- ❑ `file.tar.bz2`
- ❑ `file.tar.lzma`

不同的tar选项可以用来指定不同的压缩格式：

- ❑ -j指定bunzip2格式；
- ❑ -z指定gzip格式；
- ❑ --lzma指定lzma格式。

这些格式会在随后专门讲解压缩技术的攻略中讨论。

也可以不明确指定上面那些特定的选项来使用压缩功能。tar能够通过查看输出或输入文件的扩展名来进行压缩。为了让tar支持根据扩展名自动进行压缩，使用 -a或 --auto-compress 选项：

```
$ tar acvf archive.tar.gz filea fileb filec
filea
fileb
filec
$ tar tf archive.tar.gz
filea
fileb
filec
```

9. 从归档中排除部分文件

通过指定模式可以从归档中排除部分文件。用 --exclude [PATTERN] 排除匹配通配符样式的文件。

例如，排除所有的.txt文件：

```
$ tar -cf arch.tar * --exclude "*.txt"
```

> 样式应该使用双引号来引用，避免shell对其进行扩展。

也可以将需要排除的文件列表放入文件中，同时配合选项 -X：

```
$ cat list
filea
fileb

$ tar -cf arch.tar * -X list
```

这样就无需对filea和fileb进行归档了。

10. 排除版本控制目录

我们通常使用tar归档文件来分发源代码。大多数源代码都是使用版本控制系统进行维护的，如subversion、Git、mercurial、cvs等。版本控制系统中的代码目录包含用来管理版本的特殊目录，如.svn或.git。但源代码本身并不需要这些目录，所以不应该将它们包含在源代码的tar归档文件中。

为了在归档时排除版本控制相关的文件和目录，可以使用tar的 --exclude-vcs选项。例如：

```
$ tar --exclude-vcs -czvvf source_code.tar.gz eye_of_gnome_svn
```

11. 打印总字节数

有时候我们需要知道归档了多少字节。用-totals就可以在归档完成之后打印出总归档字节数：

```
$ tar -cf arc.tar * --exclude "*.txt" --totals
Total bytes written: 20480 (20KiB, 12MiB/s)
```

6.2.5　参考

6.4节会讲解gzip命令。

6.3　用 cpio 归档

cpio是另一种类似于tar的归档格式。它用来将多个文件和文件夹存储为单个文件，同时保留所有的文件属性，如权限、文件所有权等。不过，cpio并不像tar那么常用。它多用于RPM软件包（Fedora使用这种格式）、Linux内核的initramfs文件（包含了内核镜像）等。这则攻略将给出几种cpio的用法。

6.3.1　实战演练

cpio通过stdin获取输入文件名，并将归档文件写入stdout。我们必须将stdout重定向到一个文件来接收cpio的输出。

(1) 创建测试文件：

```
$ touch file1 file2 file3
```

(2) 将测试文件按照下面的方法进行归档：

```
$ echo file1 file2 file3 | cpio -ov > archive.cpio
```

(3) 列出cpio归档文件中的内容：

```
$ cpio -it < archive.cpio
```

(4) 从cpio归档文件中提取文件：

```
$ cpio -id < archive.cpio
```

6.3.2　工作原理

对于归档命令：

- -o指定了输出；
- -v用来打印归档文件列表。

> 通过cpio，我们能够利用文件的绝对路径进行归档。/usr/somedir就是一个绝对路径，因为它是以根目录（/）作为路径的起始。
>
> 相对路径不以/开头，而是以当前目录作为起始点。例如，test/file表示有一个目录test，而file位于test目录中。
>
> 当进行提取时，cpio会将内容提取到绝对路径中。而tar会移去绝对路径开头的/，将之转换为相对路径。

在列出给定cpio归档文件所有内容的命令中：

- -i用于指定输入；
- -t表示列出归档文件中的内容。

当使用命令进行提取时，-d用来表示提取。cpio在覆盖文件时不会发出提示。

6.4　使用 **gzip** 压缩数据

gzip是GNU/Linux平台下常用的压缩格式。gzip、gunzip、zcat都可以处理这种压缩文件类型。但gzip只能压缩单个文件或数据流，而无法对目录和多个文件进行归档。因此我们需要先创建tar归档文件，然后再用gzip进行压缩。让我们来看看gzip的使用方法。

6.4.1　实战演练

gzip可以用于压缩与解压缩。

(1) 要使用gzip压缩文件，可以使用下面的命令：

```
$ gzip filename
$ ls
filename.gz
```

(2) 将gzip文件解压缩的方法如下：

```
$ gunzip filename.gz
$ ls
file
```

(3) 列出压缩文件的属性信息：

```
$ gzip -l test.txt.gz
compressed        uncompressed  ratio uncompressed_name
  35                     6 -33.3% test.txt
```

(4) gzip命令可以从stdin中读入文件，也可以将压缩文件写出到stdout。

从stdin读入并将压缩后的数据写出到stdout：

```
$ cat file | gzip -c > file.gz
```

选项 -c用来将输出指定到stdout。

(5) 我们可以指定gzip的压缩级别。用 --fast或 --best选项分别提供最低或最高的压缩比。

6.4.2　补充内容

gzip命令通常与其他命令结合使用。它还有一些高级选项可以用来指定压缩比。让我们来看看gzip的这些特性。

1. 压缩归档文件

我们通常将gzip与归档文件结合使用。有两种方法可以创建经由gzip压缩过的归档文件。

- 方法 1

```
$ tar -czvvf archive.tar.gz [FILES]
```

或者

```
$ tar -cavvf archive.tar.gz [FILES]
```

选项 -a表明从文件扩展名自动推断压缩格式。

- 方法 2

首先，创建一个tar归档文件：

```
$ tar -cvvf archive.tar [FILES]
```

压缩tar归档文件：

```
$ gzip archive.tar
```

如果有大量文件（上百个）需要归档及压缩，我们可以采用方法2，并稍作变动。将多个文件作为命令行参数传递给tar的问题在于：tar只能从命令行中接受有限个文件。要解决这个问题，我们可以在循环中使用追加选项（-r）来逐个添加文件：

```
FILE_LIST="file1  file2  file3  file4  file5"

for f in $FILE_LIST;
do
tar -rvf archive.tar $f
done

gzip archive.tar
```

如果要提取经由gzip压缩的归档文件中的内容，可以使用：

```
$ tar -xavvf archive.tar.gz -C extract_directory
```

在上面的命令中，选项-a用于自动检测压缩格式。

2. zcat——无需解压缩，直接读取gzip格式文件

zcat命令无需人工干涉，直接就可以.gz文件中的内容提取到stdout。下面的方法可以在保持.gz文件不变的情况下，将其中的文件提取到stdout中：

```
$ ls
test.gz

$ zcat test.gz
A test file
# 文件test包含了一行文本"A test file"

$ ls
test.gz
```

3. 压缩率

我们可以指定压缩率，它共有9级，其中：

6

- 1级的压缩率最低，但是压缩速度最快；
- 9级的压缩率最高，但是压缩速度最慢。

你可以按照下面的方法指定压缩比：

```
$ gzip -5 test.img
```

这应该能在压缩速度和压缩比之间获得一个不错的平衡。

4. 使用bzip2

bzip2是另一种常用的工具，其功能和语法同gzip非常类似。唯一的不同在于bzip2的压缩效率比gzip更高，但花费的时间比gzip更长。

用bzip2进行压缩：

```
$ bzip2 filename
```

解压缩bzip2格式的文件：

```
$ bunzip2 filename.bz2
```

生成tar.bz2文件并从中提取内容的方法同之前介绍的tar.gz类似：

```
$ tar -xjvf archive.tar.bz2
```

其中-j表明该归档文件是bzip2格式。

5. 使用lzma

lzma是另一种压缩工具，它的压缩率甚至比gzip和bzip2更好。

使用lzma进行压缩：

```
$ lzma filename
```

解压缩lzma文件：

```
$ unlzma filename.lzma
```

可以使用tar命令的--lzma选项对生成的tar归档文件进行压缩或提取：

```
$ tar -cvvf --lzma archive.tar.lzma [FILES]
```

或者

```
$ tar -cavvf archive.tar.lzma [FILES]
```

如果要将经过lzma压缩过的tar归档文件中的内容提取到指定的目录中，可以使用：

```
$ tar -xvvf --lzma archive.tar.lzma -C extract_directory
```

其中，-x用于提取内容，--lzma指定使用lzma对归档文件进行解压缩。

我们也可以用：

```
$ tar -xavvf archive.tar.lzma -C extract_directory
```

6.4.3　参考

6.2节讲解了tar命令。

6.5　用 zip 归档和压缩

ZIP作为一种流行的压缩格式，在很多平台中都可以看到它的身影。在Linux下，它的应用不如gzip或bzip2那么广泛，但是Internet上的文件通常都采用这种格式。在这这攻略中，我们会看到如何使用zip进行压缩与解压缩。

6.5.1　实战演练

来看看zip的各种选项的用法。

(1) 对归档文件采用ZIP格式进行压缩：

```
$ zip archive_name.zip [SOURCE FILES/DIRS]
```

例如：

```
$ zip file.zip file
```

该命令会生成file.zip。

(2) 对目录和文件进行递归操作：

```
$ zip -r archive.zip folder1 folder2
```

其中，-r用于指定递归操作。

(3) 要从ZIP文件中提取内容，可以使用：

```
$ unzip file.zip
```

在完成提取操作之后，unzip并不会删除file.zip（这一点与unlzma和gunzip不同）。

(1) 如果需要更新压缩文件中的内容，使用选项 -u：

```
$ zip file.zip -u newfile
```

(2) 从压缩文件中删除内容，则使用-d：

```
$ zip -d arc.zip file.txt
```

(3) 列出压缩文件中的内容：

```
$ unzip -l archive.zip
```

6.5.2　工作原理

尽管同大多数我们已经讲过的归档、压缩工具类似，但zip在完成归档之后并不会删除源文件，这一点与lzma，gzip，bzip2不同。最重要的是，尽管与tar相像，zip既可以进行归档，也可以进行压缩，而单凭tar，则无法进行压缩操作。

6.6　更快速的归档工具 pbzip2

如今大多数计算机都配备了至少两个处理器核心，这基本上相当于拥有了两块物理CPU。但是仅仅是一块多核CPU并不代表程序可以运行得更快，重要的是程序自身能够利用多个处理器核心来提高运行速度。

我们目前已经看到的多数压缩命令只能利用单个处理器核心，所以速度并不会特别快。而pbzip2能够借助多个处理器核心来降低压缩文件所需的时间。

6.6.1　预备知识

多数发布版中通常都没有预装pbzip2，你得使用软件包管理器自行安装。

6.6.2　实战演练

来看看如何使用pbzip2来压缩及提取文件。

(1) 压缩单个文件：

```
pbzip2 myfile.tar
```

pbzip2会自动检测系统中处理器核心的数量，然后将myfile.tar压缩成myfile.tar.bz2。

(2) 要将多个文件或目录进行归档及压缩，可以使用tar配合pbzip2来实现：

```
tar cf myfile.tar.bz2 --use-compress-prog=pbzip2 dir_to_compress/
```

或者

```
tar -c directory_to_compress/ | pbzip2 -c > myfile.tar.bz2
```

(3) 从pbzip2格式的文件中进行提取。

如果是tar.bz2文件，我们可以一次性完成解压缩和提取工作：

```
pbzip2 -dc myfile.tar.bz2 | tar x
```

如果是经过pbzip2压缩过的归档文件，可以使用：

```
pbzip2 -d myfile.tar.bz2
```

6.6.3 工作原理

pbzip2在内部使用的压缩算法和bzip2一样，但是它利用pthreads（一个线程库）来同时对多个数据块进行压缩。不过这一切对于用户而言都是透明的，结果就是更快的压缩速度。

同gzip或bzip2一样，pbzip2并不能创建归档文件，它只能对单个文件进行操作。如果要压缩多个文件或目录，还得结合tar来使用。

6.6.4 补充内容

pbzip2还有另外一些有用的选项。

1. 手动指定处理器数量

使用pbzip2的-p选项来手动指定处理器核心的数量。如果无法自动检测处理器核心数量或是希望能够释放一些处理核心来进行其他工作，-p选项就能派上用场了。

```
pbzip2 -p4 myfile.tar
```

上面的命令告诉pbzip2使用4个处理器核心。

2. 指定压缩比

像其他压缩工具一样，我们可以使用从1到9的选项来分别指定最快和最优的压缩比。

6.7 创建压缩文件系统

squashfs是一种具有超高压缩率的只读型文件系统，这种文件系统能够将2GB~3GB的数据压缩成一个700MB的文件。Linux LiveCD（或是LiveUSB）就是使用squashfs创建的。这类CD利用只读型的压缩文件系统将根文件系统保存在一个压缩文件中。可以使用环回方式将其挂载并

装入完整的Linux环境。如果进程需要某些文件，可以将它们解压，然后载入内存中使用。

如果即需要采用超高的文件压缩率，又希望在无需解压的情况下读取少量文件，那么squashfs就能够大显身手了。解压体积较大的压缩文件可得花上一阵工夫。但如果将文件以环回形式挂载，那速度会变得飞快。因为只有出现访问请求时，对应的那部分压缩文件才会被解压缩。让我们来看看如何使用squashfs。

6.7.1　预备知识

squashfs在内部采用了gzip和lzma这类压缩算法。所有的现代Linux发行版都对其提供了支持。但要想创建squashfs文件，则需要额外安装squshfs-tools。

6.7.2　实战演练

来看看如何创建并挂载squashfs文件。

(1) 添加源目录和文件，创建一个squashfs文件：

```
$ mksquashfs SOURCES compressedfs.squashfs
```

SOURCES部分可以是通配符或文件、目录路径。

例如：

```
$ sudo mksquashfs /etc test.squashfs
Parallel mksquashfs: Using 2 processors
Creating 4.0 filesystem on test.squashfs, block size 131072.
[=======================================] 1867/1867 100%
```

> 还有更多的细节信息会出现在终端上。由于版面的限制，这里就不再列出这些信息了。

(2) 利用环回形式挂载squashfs文件：

```
# mkdir /mnt/squash
# mount -o loop compressedfs.squashfs /mnt/squash
```

你可以访问/mnt/squashfs访问其中的内容。

6.7.3　补充内容

可以指定额外的参数来定制squashfs文件系统。让我们来看看这些命令选项。

在创建squashfs文件时排除部分文件

创建squashfs文件时，我们可以排除部分文件。这些文件可以用文件列表或通配符来指定。

使用选项-e，将需要排除的文件列表以命令行参数的方式来指定。例如：

```
$ sudo mksquashfs /etc test.squashfs -e /etc/passwd /etc/shadow
```

其中，选项-e用于将文件passwd和shadow排除在外。

也可以将需要排除的文件名列表写入文件，然后用 -ef指定该文件：

```
$ cat excludelist
/etc/passwd
/etc/shadow

$ sudo mksquashfs /etc test.squashfs -ef excludelist
```

如果希望在排除文件列表中使用通配符，那么可以使用 -wildcard选项。

6.8 使用 rsync 备份系统快照

备份数据算得上是多数系统管理员日常必备的工作。除了备份本地文件，我们可能还得对Web服务器或远端数据进行备份。rsync可以对位于不同位置的文件和目录进行同步，它利用差异计算以及压缩技术来最小化数据传输量。相对于cp命令，rsync的优势在于使用了高效的差异算法。另外，它还支持网络数据传输。在进行复制的同时，rsync会比较源端和目的端的文件，只有当文件有更新时才进行复制。rsync也支持压缩、加密等多种特性。让我们看看rsync的使用方法。

6

6.8.1 实战演练

下面来看看如何使用rsync进行文件复制及备份。

(1) 将源目录复制到目的端：

```
$ rsync -av source_path destination_path
```

例如：

```
$ rsync -av /home/slynux/data slynux@192.168.0.6:/home/backups/data
```

其中：

- -a表示要进行归档；
- -v表示在stdout上打印出细节信息或进度。

上面的命令会以递归的方式将所有的文件从源路径复制到目的路径。我们可以指定

本地路径，也可以指定远端路径。

(2) 将数据备份到远程服务器或主机：

```
$ rsync -av source_dir username@host:PATH
```

如果需要在目的端建立一份镜像，只需要定期运行同样的rsync命令即可。它只会对更改过的文件进行复制。

(3) 用下面的方法将远程主机上的数据恢复到本地主机：

```
$ rsync -av username@host:PATH destination
```

> rsync命令用SSH连接远程主机，因此必须使用user@host这种形式设定远程主机的地址，其中user代表用户名，host代表远程主机的IP地址或主机名。而PATH指定需要从中复制数据的远程主机上的路径。
>
> 确保远程主机上安装并运行着OpenSSH服务器。如果连接远程主机时不希望输入密码，可以参考7.8节。

(4) 通过网络进行传输时，压缩数据能够明显改善传输效率。我们可以用rsync的选项 -z 指定在网络传输时压缩数据。例如：

```
$ rsync -avz source destination
```

(5) 将一个目录中的内容同步到另一个目录：

```
$ rsync -av /home/test/ /home/backups
```

这条命令将源目录（/home/test）中的内容（不包括目录本身）复制到现有的backups目录中。

(6) 将包括目录本身在内的内容复制到另一个目录中：

```
$ rsync -av /home/test /home/backups
```

这条命令将包括源目录本身（/home/test）在内的内容复制到新创建的目录backups中。

> 就路径格式而言，如果我们在源路径末尾使用/，那么rsync会将sourch_path尾端目录中的所有内容复制到目的端。
>
> 如果没有使用/，rsync会将sourch_path尾端目录本身复制到目的端。

以下命令复制test目录到目的端目录中：

```
$ rsync -av /home/test /home/backups/
$ rsync -av /home/test /home/backups
```

> 如果在destination_path末尾使用/，那么rsync会将来自源端的内容复制到目的端目录中。
>
> 如果没有使用/，rsync会在目的端路径尾部创建一个同名目录，然后将源端内容复制到这个目录中。

6.8.2 工作原理

rsync所使用的源路径和目的路径既可以是本地路径，也可以是远程路径。最重要的是，两者皆可以是远程路径。通常使用SSH进行远程连接，由rsync来决定应该复制哪些文件。本地路径和远程路径看起来像这样：

- /home/slynux/data（本地路径）
- slynux@192.168.0.6:/home/backups/data（远程路径）

/home/slynux/data指定的是执行rsync命令的那台主机上的绝对路径。slynux@192.168.0.6:/home/backups/data指定的是IP地址为192.168.0.6的主机，以用户slynux的身份登录，路径为/home/backups/data。

6.8.3 补充内容

可以通过命令行选项来指定rsync命令的其他功能。让我们来逐项查看。

1. 在使用rsync进行归档的过程中排除部分文件

对远端内容进行归档时，有些文件并不需要进行更新。我们可以告知rsync将某些文件从本次操作中排除。有两个选项可以用来排除文件：

```
--exclude PATTERN
```

可以通过通配符指定需要排除的文件。例如：

```
$ rsync -avz /home/code/some_code /mnt/disk/backup/code --exclude "*.txt"
```

该命令不对.txt文件进行备份。

或者我们可以通过一个列表文件指定需要排除的文件。

这可以利用--exclude-from FILEPATH。

2. 在更新rsync备份时，删除不存在的文件

默认情况下，rsync并不会在目的端删除那些在源端已不存在的文件。如果要删除这些已不存在文件，使用rsync的 --delete选项：

```
$ rsync -avz SOURCE DESTINATION --delete
```

3. 定期进行备份

你可以创建一个cron任务来定期进行备份。

下面是一个简单的例子：

```
$ crontab -ev
```

添加上这么一行：

```
0 */10 * * * rsync -avz /home/code user@IP_ADDRESS:/home/backups
```

上面的crontab条目将rsync调度为每10个小时运行一次。

/10处于crontab语法中的钟点位（hour position），/10表明每10小时执行一次备份。如果/10出现在分钟位（minutes postion），那就是每10分钟执行一次备份。

请参阅9.7节了解如何配置crontab。

6.9 用 Git 进行基于版本控制的备份

人们在备份数据时会采取不同的策略。不过比起将整个源目录复制到备份目录中，并使用日期或时间作为版本号，利用差异备份要更有效，因为前者只会白白浪费存储空间。我们只需要复制那些在备份之后发生变化的那些文件，这叫做增量备份。可以用rsync这类工具来手动创建增量备份，不过恢复这种备份可不是件易事。维护和恢复变更的最好方法是使用版本控制系统。由于代码的变更相当频繁，版本控制系统多用于软件开发与代码维护中。Git是目前最具声望、最高效的版本控制系统。我们可以在非编程环境下用Git进行常规文件的备份。

6.9.1 预备知识

我们有一个包含了多个文件和子目录的目录。现在需要跟踪目录内容的变更并对其进行备份。如果数据受损或丢失，必须能够恢复数据之前的备份。我们需要将数据备份到本地主机或远程主机中。使用软件包管理器安装Git，然后来学习它的用法。

6.9.2 实战演练

先来看看如何使用Git进行数据备份。

(1) 进入需要备份的目录：

```
$ cd /home/data/source
```

将其作为需要被跟踪的目录源。

(2) 设置并初始化远端备份目录。在远程主机中创建备份目录：

```
$ mkdir -p /home/backups/backup.git
$ cd /home/backups/backup.git
$ git init --bare
```

在源主机中执行下列步骤。

(1) 在源主机中将用户详细信息添加到Git：

```
$ git config --global user.name  "Sarath Lakshman"
$ git config --global user.email slynux@slynux.com
```

(2) 初始化主机中需要进行备份的源目录。在源目录中执行下列命令：

```
$ git init
Initialized empty Git repository in /home/backups/backup.git/

$ git commit --allow-empty -am "Init"
[master (root-commit) b595488] Init
```

(3) 在源目录中执行下列命令来添加远程Git目录并同步备份：

```
$ git remote add origin user@remotehost:/home/backups/backup.git

$ git push origin master
Counting objects: 2, done.
Writing objects: 100% (2/2), 153 bytes, done.
Total 2 (delta 0), reused 0 (delta 0)
To user@remotehost:/home/backups/backup.git
 * [new branch]      master -> master
```

(4) 为Git跟踪添加或删除文件。

下面的命令将当前目录下的所有文件和文件夹添加到备份列表中：

```
$ git add *
```

我们可以有条件地添加某些文件到备份列表中：

```
$ git add *.txt
$ git add *.py
```

删除不需要跟踪的文件和文件夹：

```
$ git rm file
```

也可以使用通配符：

```
$ git rm *.txt
```

(5) 检查点或标注备份点。

用下列命令来标注一个附带消息的备份的检查点：

```
$ git commit -m "Commit Message"
```

我们需要定期更新远端的备份，因此得设置一个cron任务（例如，每5个小时进行一次备份）：

创建一个crontab条目：

```
0 */5 * * *  /home/data/backup.sh
```

创建脚本/home/data/backup.sh：

```
#!/bin/ bash
cd /home/data/source
git add .
git commit -am "Backup taken at @ $(date)"
git push
```

现在我们就算是完成了备份系统的设置。

(6) 查看所有版本的备份：

```
$ git log
```

(7) 要恢复到之前的某个状态或版本，需要查看一个由32位十六进制串组成的提交ID。通过git checkout来使用提交ID。

如果提交ID为 3131f9661ec1739f72c213ec5769bc0abefa85a9，那就是：

```
$ git checkout 3131f9661ec1739f72c213ec5769bc0abefa85a9
```

以下命令可以使转化持久化：

```
$ git commit -am "Restore @ $(date) commit ID: 3131f9661ec1739f72c213ec5769
bc0abefa85a9"
```

再次查看有关版本的细节信息：

```
$ git log
```

(8) 如果工作目录由于某些原因受到了损坏，我们需要用远端的备份来进行修复。按照下面的方法从远端备份中重建损坏的内容：

```
$ git clone user@remotehost:/home/backups/backup.git
```

这条命令将创建一个包含之前全部内容的目录。

> 尽管就保留版本化的文本文件（包括文档、源代码等等）副本而言，Git表现的算是相当不错了。但最好不要将其用于大量的二进制数据备份。例如用Git来对图片库进行备份/版本管理，就没什么意义了。这是因为如果涉及的是二进制文件，Git保留的是整个文件，而非文件之间的差异，这样会占用大量的磁盘空间。

6.10 用 **fsarchiver** 创建全盘镜像

`fsarchiver`可以将整个文件系统中的内容保存成一个压缩形式的归档文件。鉴于这种能力，它成为功能最为完备，也最为易用的备份工具之一。

`fsarchiver`作为`partimage`（一款知名的**文件系统**备份工具）的接班人，能够支持像ext4这种较新的文件系统，不过`partimage`配备了一套小型的图形用户界面，相比之下更易用一些。

6.10.1 预备知识

`fsarchiver`默认并没有安装在大多数发布版中。你得用软件包管理器自行安装。更多的信息可以参考http://www.fsarchiver.org/Installation。

6.10.2 实战演练

(1) 创建文件系统/分区备份。

使用`fsarchiver`的`savefs`选项：

```
fsarchiver savefs backup.fsa /dev/sda1
```

backup.fsa是最终的备份文件，/dev/sda/是要备份的分区。

(2) 同时备份多个分区。

还是使用savefs选项，将分区作为fsarchiver最后的参数：

```
fsarchiver savefs backup.fsa /dev/sda1 /dev/sda2
```

(3) 从备份归档中恢复分区。

使用fsarchiver的restfs选项：

```
fsarchiver restfs backup.fsa id=0,dest=/dev/sda1
```

id=0表明我们希望从备份归档中提取第一个分区的内容，将其恢复到由dest=/dev/sda1所指定的分区中。

(4) 从备份归档中恢复多个分区。

像之前一样，使用restfs选项：

```
fsarchiver restfs backup.fsa id=0,dest=/dev/sda1 id=1,dest=/dev/sdb1
```

我们使用了两组id，dest告诉fsarchiver从备份中将前两个分区的内容恢复到指定的物理分区中。

6.10.3　工作原理

和tar的工作原理非常类似，fsarchiver遍历整个文件系统来生成一个文件列表，然后将所有的文件保存在压缩过的归档文件中。不像tar那样只保存文件信息，fsarchiver还要对文件系统进行备份。这意味着它可以很容易地将备份恢复到一个全新的系统中，因为不必再重建文件系统了。

如果你是第一次看到/dev/sda1这样的分区记法，那有必要解释一下。在Linux中，/dev下存放的都是称为设备文件的一类特殊文件，它们分别指向某个物理设备。sda1中的sd指的是SATA disk，接下来的字母可以是a，b，c等，最后跟上分区编号，如图6-1所示。

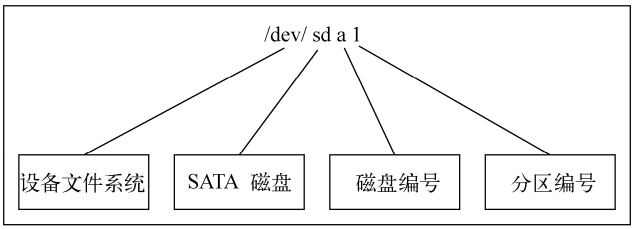


图6-1　显示了Linux中磁盘设备的文件名的各个组成部分

第 7 章

无网不利

本章内容

- 网络设置
- 使用ping
- 列出网络上所有的活动主机
- 用SSH在远程主机上运行命令
- 通过网络传输文件
- 连接无线网络
- 用SSH实现无密码自动登录
- 使用SSH进行端口转发
- 在本地挂载点上挂载远程驱动器
- 网络流量与端口分析
- 创建套接字
- 互联网连接共享
- 使用iptables架设简易防火墙

7.1 简介

联网就是将主机进行互联以形成网络，使得网络中的主机得以交换信息。应用最广泛的网站栈就是TCP/IP，其中每个节点都分配了一个用作标识的独一的IP地址。有很多联网参数，如子网掩码、路由、端口和DNS等，我们需要对这些知识有一个基本的认识。

一些利用网络的应用通过打开并连接到端口来进行运作，端口用于指代某种服务，如数据传输、远程shell登录等。有些管理任务可以通过网络进行。shell脚本也能用来配置网络节点、测试主机是否可用、自动执行远程主机命令等。本章着重介绍网络相关的工具和命令，以及如何用它们解决各种问题。

7.2 网络设置

在深入学习与联网相关的攻略之前，有必要简单了解一下网络设置、相关术语以及用于分配IP地址、添加路由等命令。这则攻略会从头开始介绍GNU/Linux中用于联网的各种命令及用法。

7.2.1　预备知识

网络接口用于将主机连接到网络。在Linux中通常使用eth0、eth1（指代以太网接口）这种方式来命名网络接口。还有一些其他的接口，如usb0、wlan0等，分别对应USB网络接口、无线LAN。

在这则攻略中会涉及如下命令：ifconfig、route、nslookup和host。

ifconfig命令用于配置及显示网络接口、子网掩码等详细信息。它通常位于/sbin/ifconfig中。

7.2.2　实战演练

(1) 列出当前的网络接口配置：

```
$ ifconfig
lo        Link encap:Local Loopback
inet addr:127.0.0.1  Mask:255.0.0.0
inet6addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:6078 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6078 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
          RX bytes:634520 (634.5 KB)  TX bytes:634520 (634.5 KB)
wlan0     Link encap:EthernetHWaddr 00:1c:bf:87:25:d2
inet addr:192.168.0.82  Bcast:192.168.3.255  Mask:255.255.252.0
inet6addr: fe80::21c:bfff:fe87:25d2/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:420917 errors:0 dropped:0 overruns:0 frame:0
          TX packets:86820 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
          RX bytes:98027420 (98.0 MB)  TX bytes:22602672 (22.6 MB)
```

ifconfig输出的最左边一列是网络接口名，右边的若干列显示对应的网络接口的详细信息。

(2) 手动设置网络接口的IP地址：

```
# ifconfig wlan0 192.168.0.80
```

你需要使用root账号运行上述命令。192.168.0.80是要设置的IP地址。

使用以下命令设置比IP地址的子网掩码：

```
# ifconfig wlan0 192.168.0.80  netmask 255.255.252.0
```

(3) 自动配置网络接口。如果你连接到一个支持自动分配IP的有线网络中，只需要使用下面的命令就可以配置网络接口了：

```
# dhclient eth0
```

7.2.3 补充内容

让我们再看一些其他的基础命令以及用法。

1. 打印网络接口列表

这个单行命令可以打印系统可用的网络接口列表。

```
$ ifconfig | cut -c-10 | tr -d ' ' | tr -s '\n'
lo
wlan0
```

ifconfig输出的前10个字符是用于保留打印网络接口的名称，因此我们用cut命令提取每一行的前10个字符。tr -d ' '删除每一行的所有空格。用tr -s '\n'压缩重复的换行符生成接口名称列表。

2.显示IP地址

ifconfig会显示系统中所有可用网络接口的详细信息。我们可以限制它只显示某个特定的接口信息：

```
$ ifconfig iface_name
```

例如：

```
$ ifconfig wlan0
wlan0     Link encap:EthernetHWaddr 00:1c:bf:87:25:d2
inet addr:192.168.0.82  Bcast:192.168.3.255
    Mask:255.255.252.0
```

在上面命令输出中，我们感兴趣的是IP地址、广播地址、硬件地址和子网掩码。它们分别为：

- HWaddr 00:1c:bf:87:25:d2是硬件地址（MAC地址）；
- inet addr:192.168.0.82是IP地址；
- Bcast:192.168.3.255是广播地址；
- Mask:255.255.252.0是子网掩码。

在编写某些脚本时，我们可能需要在脚本中提取某些地址来做进一步的处理。常见的是提取IP地址。要从ifconfig输出中提取IP地址，可以使用：

```
$ ifconfig wlan0 | egrep -o "inet addr:[^ ]*" | grep -o "[0-9.]*"
192.168.0.82
```

第一条命令egrep -o "inet addr:[^]*" 会打印出inet addr:192.168.0.82。模式以inet addr:作为起始，以非空格字符序列（由 [^]* 指定）作为结束。在接下来的管道操作中，打印出数字与点号（.）的组合。

3. 硬件地址（MAC地址）欺骗

在某些情况下，需要利用硬件地址对网络上的计算机进行认证或过滤，对此我们可以使用硬件地址欺骗。硬件地址在ifconfig输出中是以HWaddr 00:1c:bf:87:25:d2的形式出现的。

我们能够按照下面的方法在软件层面上进行硬件地址欺骗：

```
# ifconfig eth0 hw ether 00:1c:bf:87:25:d5
```

在上面的命令中，00:1c:bf:87:25:d5是分配的新MAC地址。如果我们需要通过部署了MAC认证的Internet服务提供商才能够访问Internet，这招就能排上用场了。注意，这在机器重启之后就失效了。

4. 名字服务器与DNS（域名服务）

互联网最根本的寻址方案是IP地址（采用点分十进制形式，例如202.11.32.75）。然而Internet上的资源（比如网站）是通过被称为URL或域名的ASCII字符组合来访问的，例如google.com就是一个域名，它对应一个（或多个）IP地址。在浏览器中输入IP地址同样可以访问www.google.com。

这种利用符号名对IP地址进行抽象的技术就被称为**域名服务**（DNS）。当我们输入google.com，计算机使用配置好的DNS服务器将域名解析为对应的IP地址。在本地网络中，我们可以设置本地DNS，以便使用主机名来命名本地网络上的主机。

分配给当前系统的名字服务器可以通过读取 /etc/resolv.conf查看。例如：

```
$ cat /etc/resolv.conf
nameserver 8.8.8.8
```

我们可以像下面这样手动添加名字服务器：

```
# echo nameserver IP_ADDRESS >> /etc/resolv.conf
```

该如何获得域名所对应的IP地址呢？获取IP地址最简单的方法就是ping给定的域名，然后查看回应信息。例如：

```
$ ping google.com
PING google.com (64.233.181.106) 56(84) bytes of data.
#64.233.181.106是google.com对应的IP地址。
```

一个域名可以分配多个IP地址。对于这种情况，ping只会显示其中的一个地址。要想获取分配给域名的所有IP地址，就得使用DNS查找工具了。

5. DNS查找

有多种基于命令行的DNS查找工具，这些工具会向DNS服务器请求解析IP地址。host和nslookup就是此类DNS查找工具中的两个。

当执行host时，它会列出某个域名所有的IP地址。nslookup类似于host命令，它用于查询DNS相关的细节信息以及名字解析。例如：

```
$ host google.com
google.com has address 64.233.181.105
google.com has address 64.233.181.99
google.com has address 64.233.181.147
google.com has address 64.233.181.106
google.com has address 64.233.181.103
google.com has address 64.233.181.104
```

host也可以列出DNS资源记录（DNS resource record）：

```
$ nslookup google.com
Server:    8.8.8.8
Address:  8.8.8.8#53

Non-authoritative answer:
Name:  google.com
Address: 64.233.181.105
Name:  google.com
Address: 64.233.181.99
Name:  google.com
Address: 64.233.181.147
Name:  google.com
Address: 64.233.181.106
Name:  google.com
Address: 64.233.181.103
Name:  google.com
Address: 64.233.181.104

Server:    8.8.8.8
```

上面最后一行对应着用于DNS解析的默认名字服务器。

如果不使用DNS服务器，也可以为IP地址解析添加符号名，这只需要向文件 /etc/hosts中加入条目即可。

7

用下面的方法进行添加：

```
# echo IP_ADDRESS symbolic_name >> /etc/hosts
```

例如：

```
# echo 192.168.0.9 backupserver >> /etc/hosts
```

添加了条目之后，任何时候解析backupserver，都会返回192.168.0.9。

6. 显示路由表信息

多个网络相互连接是很常见的场景。例如在大学里，不同的职能部门可能处于一个网络中。在这种情况下，一个网络中的设备如果想同另一个网络中的设备进行通信，就需要借助某个同时连接了两个网络的设备。这个特殊的设备被称为**网关**，它的作用是在不同的网络中转发分组。

操作系统维护着一个叫做**路由表的表格**，它包含了关于分组如何转发以及通过网络中的哪些节点转发的信息。可以用下面的方法显示路由表：

```
$ route
Kernel IP routing table
Destination      Gateway   Genmask       Flags  Metric   Ref   UseIface
192.168.0.0         *      255.255.252.0  U      2       0      0wlan0
link-local          *      255.255.0.0    U      1000    0      0wlan0
default          p4.local  0.0.0.0        UG     0       0      0wlan0
```

也可以使用：

```
$ route -n
Kernel IP routing table
Destination    Gateway       Genmask        Flags Metric Ref   Use    Iface
192.168.0.0    0.0.0.0       255.255.252.0   U      2     0      0    wlan0
169.254.0.0    0.0.0.0       255.255.0.0     U      1000  0      0    wlan0
0.0.0.0        192.168.0.4   0.0.0.0         UG     0     0      0    wlan0
```

-n指定以数字形式显示地址。如果使用-n，route会以数字形式的IP地址显示每一个条目；否则，如果IP地址具有对应的DNS条目，就会显示符号形式的主机名。

设置默认网关：

```
# route add default gw IP_ADDRESS INTERFACE_NAME
```

例如：

```
# route add default gw 192.168.0.1 wlan0
```

7.2.4　参考

- 1.3节讲解了PATH变量。
- 4.3节讲解了grep命令。

7.3　使用 **ping**

ping是每位用户都应该首先了解的最基础的网络命令。绝大多数操作系统上都包含了该命

令。ping也是一个验证网络上两台主机连通性的诊断工具，能够找出网络上的活动主机。让我们看看该命令的用法。

7.3.1　实战演练

为了检查网络上两台主机之间的连通性，ping命令使用**互联网控制消息协议**（Internet Control Message Protocol，ICMP）中的echo分组。当向某台主机发送echo分组时，如果分组能够送达且该主机处于活动状态，那么它就会返回一条回应。

检查某台主机是否可以到达：

```
$ ping ADDRESS
```

ADDRESS可以是主机名、域名或者IP地址。

ping会连续发送分组，回应信息将被打印在终端上。用Ctrl+C来停止ping命令。

- 如果主机可以到达，那么输出信息如下所示：

```
$ ping 192.168.0.1
PING 192.168.0.1 (192.168.0.1) 56(84) bytes of data.
64 bytes from 192.168.0.1: icmp_seq=1 ttl=64 time=1.44 ms
^C
--- 192.168.0.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.440/1.440/1.440/0.000 ms

$ ping google.com
PING google.com (209.85.153.104) 56(84) bytes of data.
64 bytes from bom01s01-in-f104.1e100.net (209.85.153.104): icmp_seq=1 ttl=53
time=123 ms
^C
--- google.com ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 123.388/123.388/123.388/0.000 ms
```


- 如果主机不可到达，则输出如下所示：

```
$ ping 192.168.0.99
PING 192.168.0.99 (192.168.0.99) 56(84) bytes of data.
From 192.168.0.82 icmp_seq=1 Destination Host Unreachable
From 192.168.0.82 icmp_seq=2 Destination Host Unreachable
```

如果主机不可到达，ping返回错误信息“Destination Host Unreachable”。

> 网络管理员通常会对路由器进行配置，使其不响应ping命令。这样做是为了降低安全风险，因为ping可以被攻击者（使用蛮力）用来获取主机的IP地址。

7.3.2 补充内容

除了检查网络上两点之间的连通性，ping命令还可以通过其他选项来获取有用信息。让我们看看ping的其他选项。

1. 往返时间

ping命令可以用来得出网络上两台主机之间的**往返时间**（Round Trip Time，RTT）。它是分组从源主机到目的主机一来一回的时间。RTT的单位是毫秒，该时间可以从ping命令中获知。例如：

```
--- google.com ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4000ms
rtt min/avg/max/mdev = 118.012/206.630/347.186/77.713 ms
```

其中，最小的RTT是118.012ms，平均RTT是206.630ms，最大RTT是347.186ms。ping输出中的mdev（77.713ms）代表平均偏差（mean deviation）。

2. 限制发送的分组数量

ping命令会不停地发送echo分组并等待回复，直到按下Ctrl+C为止。我们可以用选项 -c限制所发送的echo分组的数量。用法如下：

```
-c COUNT
```

例如：

```
$ ping 192.168.0.1 -c 2
PING 192.168.0.1 (192.168.0.1) 56(84) bytes of data.
64 bytes from 192.168.0.1: icmp_seq=1 ttl=64 time=4.02 ms
64 bytes from 192.168.0.1: icmp_seq=2 ttl=64 time=1.03 ms

--- 192.168.0.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 1.039/2.533/4.028/1.495 ms
```

在上面的例子中，ping命令发送了2个echo分组后就停止发送。果我们需要通过脚本ping一组IP地址来检查主机的状态，那么这个技巧就能派上用场了。

3. ping命令的返回状态

ping命令如果执行顺利，会返回退出状态0；否则，返回非0。执行顺利意味着目的主机能够到达，否则意味着目的主机不可到达。

返回状态可以通过下面的方法轻松获得：

```
$ ping domain -c2
if [ $? -eq 0 ];
then
  echo Successful ;
else
  echo Failure
fi
```

4. **Traceroute**

当应用程序通过互联网请求服务时，服务器可能位于远端，两者之间通过多个网关或设备节点相连。分组要穿过这些网关才能到达目的地。有一个很有意思的命令`traceroute`，它可以显示分组途径的所有网关的地址。`traceroute`信息可以帮助我们搞明白分组到达目的地需要经过多少跳（hop）。中途的网关或路由器的数量给出了一个测量网络上两个节点之间距离的度量（metric）。`traceroute`的输出如下：

```
$ traceroute google.com
traceroute to google.com (74.125.77.104), 30 hops max, 60 byte packets
1 gw-c6509.lxb.as5577.net (195.26.4.1) 0.313 ms 0.371 ms 0.457 ms
2 40g.lxb-fra.as5577.net (83.243.12.2) 4.684 ms 4.754 ms 4.823 ms
3 de-cix10.net.google.com (80.81.192.108) 5.312 ms 5.348 ms 5.327 ms
4 209.85.255.170 (209.85.255.170) 5.816 ms 5.791 ms 209.85.255.172
(209.85.255.172) 5.678 ms
5 209.85.250.140 (209.85.250.140) 10.126 ms 9.867 ms 10.754 ms
6 64.233.175.246 (64.233.175.246) 12.940 ms 72.14.233.114
(72.14.233.114) 13.736 ms 13.803 ms
7 72.14.239.199 (72.14.239.199) 14.618 ms 209.85.255.166
(209.85.255.166) 12.755 ms 209.85.255.143 (209.85.255.143) 13.803 ms
8 209.85.255.98 (209.85.255.98) 22.625 ms 209.85.255.110
(209.85.255.110) 14.122 ms
*
9 ew-in-f104.1e100.net (74.125.77.104) 13.061 ms 13.256 ms 13.484 ms
```


> 如今的Linux发布版中还包括了一个命令`mtr`，它类似于`traceroute`，但是能够显示实时刷新的数据。对于检查网络线路质量等问题很有帮助。

7.4　列出网络上所有的活动主机

当我们管理大型局域网时，可能需要检查网络上的其他主机是否处于活动状态。一台非活动主机可能有两种情况：要么是没有开机，要么是网络连接有问题。借助shell脚本，我们可以轻易找出并报告网络上的哪一台主机处于活动状态。让我们看看这是如何实现的。

7.4.1　预备知识

在这则攻略中，我们采用两种方法。第一种方法使用ping，第二种方法使用fping。在脚本中使用fping更容易些，而且相对于ping，它具有更多的特性。fping默认并没有包含在Linux发布版中，需要用软件包管理器手动安装。

7.4.2　实战演练

让我们看看可以找出网络上所有活动主机的shell脚本，以及实现同一目标的另一种方法。

(1) 方法1

我们可以用ping命令编写脚本来查询一组IP地址，检查它们是否处于活动状态：

```
#!/bin/bash
# 文件名：ping.sh
#用途：根据网络配置对网络地址192.168.0进行修改。

for ip in 192.168.0.{1..255} ;
do
  ping $ip -c 2 &> /dev/null ;

  if [ $? -eq 0 ];
  then
    echo $ip is alive
  fi
done
```

输出如下：

```
$ ./ping.sh
192.168.0.1 is alive
192.168.0.90 is alive
```

(2) 使用fping

我们可以用已有的命令行工具来查询网络上的主机状态：

```
$ fping -a 192.160.1/24 -g 2> /dev/null
192.168.0.1
192.168.0.90
```

或者，使用：

```
$ fping -a 192.168.0.1 192.168.0.255 -g
```

7.4.3　工作原理

在方法1中，我们用ping命令找出网络上的活动主机。这里用了一个for循环对一组IP地址进行迭代。这组IP地址依照表达式192.168.0.{1..255}来生成。像{start..end}这种记法会由shell对其进行扩展并生成一组IP地址，例如192.168.0.1、192.168.0.2、192.168.0.3等，直到192.168.0.255。

ping $ip -c 2 &> /dev/null会在每次循环中ping对应的IP地址。-c 2将发送的echo分组数量限制为2。&> /dev/null用于将stderr和stdout重定向到/dev/null，使对应的信息不会在终端上打印出来。我们用 $? 获取退出状态。如果顺利退出，退出状态为0，否则为非0。因此能够ping通的IP地址就被打印出来。

在这个脚本中，ping命令都是依次执行的。尽管所有的IP地址都是彼此独立，但由于编写的是顺序式程序（sequential program），ping命令也只能按顺序执行。每执行一次ping命令，都要经历一段延迟：发送两个echo分组，接收回应或等待回应超时。

7.4.4　补充内容

我们已经讨论过了一种找出网络中所有活动主机的方法。接下来再看一些对于已经方法的改进以及另一种全新的实现方法。

1. 并行ping

要是处理255个地址的话，累积下来延迟时间可就不短了。我们可以利用并行方式来加速ping命令的执行。要使ping命令可以并行执行，可将循环体放入( )&。( )中的命令作为子shell来运行，而&会将其置入后台。例如：

```
#!/bin/bash
# 文件名: fast_ping.sh
#用途：根据网络配置对网络地址192.168.0进行修改。

for ip in 192.168.0.{1..255} ;
do
   (
      ping $ip -c2 &> /dev/null ;

      if [ $? -eq 0 ];
      then
       echo $ip is alive
      fi
   )&
  done
wait
```

7

在for循环体中执行多个后台进程，然后结束循环并终止脚本。要想等所有子进程结束之后再终止脚本，就得使用wait命令。将wait放在脚本最后，它就会一直等到所有的子脚本进程全部结束。

2. 使用fping

第二种方法使用了另一个命令fping。它同时可以ping一组IP地址，而且响应速度非常快。fping的选项如下：

- 选项 -a指定打印出所有活动主机的IP地址；
- 选项 -u指定打印出所有无法到达的主机；
- 选项 -g指定从 "IP地址/子网掩码"记法或者"IP地址范围"记法中生成一组IP地址；

```
$ fping -a 192.160.1/24 -g
```

或者

```
$ fping -a 192.160.1 192.168.0.255 -g
```

- 2>/dev/null将由于主机无法到达所产生的错误信息打印到null设备。

  也可以采用命令行参数的方式手动指定一组IP地址，或者从stdin中接收。例如：

```
$ fping -a 192.168.0.1 192.168.0.5 192.168.0.6
# 将IP地址作为参数传递
$ fping -a < ip.list
# 从文件中传递一组IP地址
```

7.4.5　参考

- 1.6节讲解了数据重定向。
- 1.17节讲解了数字比较。

7.5　使用 SSH 在远程主机上运行命令

SSH是一个很有意思的系统管理工具，它能够让你访问远程计算机上的shell，进而执行各种命令。SSH是Secure Shell的缩写，因为它使用加密通道来传输网络数据。这则攻略介绍了在远程主机上运行命令的各种方法。

7.5.1　预备知识

GNU/Linux发布版中默认都不包括SSH，需要使用软件包管理器安装openssh-server和openssh-client软件包。SSH服务默认运行在端口22之上。

7.5.2　实战演练

(1) 连接运行SSH服务器的远程主机：

```
$ ssh username@remote_host
```

其中：

- ❑ username是远程主机上的用户；
- ❑ remote_host可以是域名或IP地址。

例如：

```
$ ssh mec@192.168.0.1
The authenticity of host '192.168.0.1 (192.168.0.1)' can't be established.
RSA key fingerprint is 2b:b4:90:79:49:0a:f1:b3:8a:db:9f:73:2d:75:d6:f9.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.0.1' (RSA) to the list of known hosts.
Password:

Last login: Fri Sep  3 05:15:21 2010 from 192.168.0.82
mec@proxy-1:~$
```

ssh采用交互方式询问用户密码，一旦认证成功，将会为用户返回一个shell。

> SSH执行指纹核对（fingerprint verification）来确保用户连接到正确的远程主机。这是为了避免**中间人攻击**（man-in-the-middle attack），在这类攻击中，攻击者试图假扮成另一台计算机。在第一次连接到服务器上时，SSH默认会存储指纹信息，在今后的连接过程中核对该指纹。

SSH服务器默认在端口22上运行。但有些SSH服务器并没有使用这个端口。针对这种情况，可以用 SSH的-p port_num来指定端口。

(2) 连接运行在端口422之上的SSH服务器：

```
$ ssh user@locahost -p 422
```

你可以在远程主机的shell中执行命令。但是在shell脚本中使用SSH时，并不需要交互式的shell，因为我们需要执行多条命令并显示或存储命令输出。

> 每次都要输入密码对于自动化脚本来说显然不实际，因此要对SSH进行无密码登录配置。7.8节将讲解具体的配置方法。

(3) 要想在远程主机中执行命令，并将命令输出显示在本地shell中，使用下面的语法：

```
$ ssh user@host 'COMMANDS'
```

例如：

```
$ ssh mec@192.168.0.1 'whoami'
mec
```

可以输入多条命令，在命令之间以分号进行分隔：

```
$ ssh user@host "command1 ; command2 ; command3"
```

例如：

```
$ ssh mec@192.168.0.1  "echo user: $(whoami);echo OS: $(uname)"
Password:
user: mec
OS: Linux
```

在这个例子中，在远程主机上执行的命令是：

```
echo user: $(whoami);
echo OS: $(uname)
```

写成一般的形式就是：

```
COMMANDS="command1; command2; command3"
$ ssh user@hostname  "$COMMANDS"
```

我们也可以在命令序列中用子shell操作符传递一个更复杂的子shell。

(4) 让我们来编写一个基于SSH的shell脚本，它用来收集一组远程主机的运行时间（uptime）。运行时间是系统加电运行的时间，`uptime`命令可以用来显示系统加电后运行了多久。

假设在`IP_LIST`中的所有系统都有一个用户`test`。

```
#!/bin/bash
# 文件名: uptime.sh
# 用途:系统运行时间监视器

IP_LIST="192.168.0.1 192.168.0.5 192.168.0.9"
USER="test"

for IP in $IP_LIST;
do
  utime=$(ssh ${USER}@${IP} uptime  | awk '{ print $3 }' )
  echo $IP uptime:  $utime
done
```

输出如下：

```
$ ./uptime.sh
192.168.0.1 uptime: 1:50,
192.168.0.5 uptime: 2:15,
192.168.0.9 uptime: 10:15,
```

7.5.3 补充内容

让我们看看ssh命令的另一些选项。

1. SSH的压缩功能

SSH协议也支持对数据进行压缩传输。当带宽有限时，这一功能很方便。用ssh命令的选项-C启用这一功能：

```
$ ssh -C user@hostname COMMANDS
```

2. 将数据重定向至远程shell命令的stdin

有时候我们需要将一些数据重定向到远程shell命令的stdin。下面是实现方法：

```
$ echo 'text' | ssh user@remote_host 'echo'
text
```

或者

```
# 将文件中的数据进行重定向
$ ssh user@remote_host 'echo'  < file
```

在远程主机上，echo打印出通过stdin接收到的数据，但这些数据却是从本地主机传递到远程shell的stdin中的。

3. 在远程主机中执行图形化命令

如果你打算使用刚才介绍过的方法来运行一些需要显示某种形式的GUI的命令，就会碰上类似于“无法显示”之类的错误。这是因为ssh shell无法连接到远程主机上的X服务器。对此，你需要像这样设置变量$DISPLAY：

```
ssh user@host "export DISPLAY=:0 ; command1; command2"""
```

这将启用远程主机上的图形化输出。如果你想在本地主机上也显示图形化输出，使用SSH的X11转发选项（forwarding option）：

```
ssh -X user@host "command1; command2"
```

该命令会在远程主机上执行，但图形化输出会显示在本地主机上。

7.5.4　参考

7.8节讲解了如何实现在不输入密码的情况下，自动登录执行远程命令。

7.6　通过网络传输文件

计算机联网的主要目的就是资源共享。在资源共享方面，使用最多的是文件共享。有多种方法可以用来在网络中传输文件。这则攻略就讨论了如何用常见的协议FTP、SFTP、RSYNC和SCP传输文件。

7.6.1　预备知识

用来在网络上传输文件的命令多数都已默认包含在安装好的Linux中。通过FTP传输文件可以使用`lftp`命令，通过SSH连接传输文件可以使用`sftp`，RSYNC使用SSH与`rsync`命令，`scp`通过SSH进行传输。

7.6.2　实战演练

文件传输协议（File Transfer Protocol，FTP）是一个古老的用于在网络主机之间传输文件的文件传输协议。我们可以用`lftp`命令访问FTP服务器来进行文件传输。只有远程主机上安装有FTP服务器才能使用FTP。很多公共网站都是用FTP共享文件，它使用端口21。

要连接FTP服务器传输文件，可以使用：

```
$ lftp username@ftphost
```

它会提示你输入密码，然后显示一个像下面那样的登录提示符：

```
lftp username@ftphost:~>
```

你可以在提示符后输入命令，如下所示。

- 用`cd directory`改变目录。
- 用`lcd`改变本地主机的目录。
- 用`mkdir`创建目录。
- 列出远程机器当前目录下的文件使用`Is`。
- 用`get filename`下载文件：

```
lftp username@ftphost:~> get filename
```

- 用`put filename`从当前目录上传文件：

```
lftp username@ftphost:~> put filename
```

- ❑ 用quit退出lftp会话。

  lftp提示符支持命令自动补全。

7.6.3　补充内容

让我们看看其他可用于网络文件传输的技术及命令。

1. FTP自动传输

ftp是另一个可用于FTP文件传输的命令。相比较而言，lftp的用法更灵活。lftp和ftp为用户启动一个交互式会话（通过显示消息来提示用户输入）。如果我们不使用交互模式，而是希望进行自动文件传输，又该怎么做呢？下面的脚本可以用来实现FTP自动传输：

```
#!/bin/bash
# 文件名: ftp.sh
# 用途: 自动化 FTP 传输
HOST='domain.com'
USER='foo'
PASSWD='password'
ftp -i -n $HOST <<EOF
user ${USER} ${PASSWD}
binary
cd /home/slynux
puttestfile.jpg
getserverfile.jpg
quit
EOF
```

上面的脚本包含下列结构：

```
<<EOF
DATA
EOF
```

这是用来通过stdin向FTP命令发送数据。1.6节中已讲解了重定向到stdin的各种方法。

在示例脚本中，ftp的选项 -i关闭用户的交互会话，user ${USER} ${PASSWD}设置用户名和密码，binary将文件模式设置为二进制模式。

2. SFTP（Secure FTP，安全FTP）

SFTP是一个类似于FTP的文件传输系统，它运行在SSH连接之上并模拟成FTP接口。它不需要远端运行FTP服务器来进行文件传输，但必须安装并运行OpenSSH服务器。SFTP是一个交互式命令，提供了命令提示符。

下面的命令可用于文件传输。对于特定主机、用户和密码的自动化FTP会话来说，命令都是一样的。

```
cd /home/slynux
put testfile.jpg
get serverfile.jpg
```

运行sftp：

```
$ sftp user@domainname
```

和lftp类似，输入quit命令可以退出sftp会话。

SSH服务器有时候并不在默认的端口22上运行。如果它在其他端口运行，我们可以在sftp中用 -oPort=PORTNO来指定端口号。例如：

```
$ sftp -oPort=422 user@slynux.org
```

-oPort应该作为sftp命令的第一个参数。

3. rsync命令

作为一款重要的命令行工具，rsync广泛用于网络文件复制及系统备份。6.8节详细讲解了rsync的用法。

4. SCP（Secure Copy Program，安全复制程序）

就传统的远程复制工具rcp而言，SCP是一项更安全的文件复制技术。文件均通过SSH加密通道进行传输。我们可以像下面这样轻松地将文件传输到远程主机：

```
$ scp filename user@remotehost:/home/path
```

该命令会提示你输入密码，可以用SSH自动登录功能来免于输入密码。7.8节会讲解SSH自动登录。因此，用scp传输文件无需特定的脚本。一旦实现了SSH自动登录，scp就可以直接执行，而不再需要提示输入密码。

命令中的remotehost可以是IP地址或域名。scp命令的格式是：

```
$ scp SOURCE DESTINATION
```

SOURCE或DESTINATION可以采用形如username@localhost:/path的格式。例如：

```
$ scp user@remotehost:/home/path/filename filename
```

上面的命令将远程主机中的文件复制到当前目录，并采用给定的文件名。

如果SSH没有运行在端口22，使用 -oPort，并采用和sftp相同的语法。

5. 用SCP进行递归复制

使用scp的 -r选项，我们可以在两台网络主机之间对文件夹进行递归复制：

```
$ scp -r /home/slynux user@remotehost:/home/backups
# 将目录/home/slynux递归复制到远程主机中
```

scp的 -p选项能够在复制文件的同时保留文件的权限和模式。

7.6.4 参考

1.6节讲解了如何用EOF实现标准输入。

7.7 连接无线网络

配置以太网很简单，因为它使用物理线缆，无需认证之类的特殊要求。但是无线LAN可能就得进行认证了，比如WEP密钥以及所要连接的无线网络的ESSID（Extended Service Set Identification，扩展服务集标识）。ESSID是无线网络的名称。让我们看看如何用脚本来连接无线网络。

7.7.1 预备知识

我们需要用ifconfig分配IP地址和子网掩码才能连接上有线网络。对于无线网络来说，还需要其他工具（如iwconfig和iwlist）来配置更多的参数。

7.7.2 实战演练

连接启用了WEP（Wried Equivalent Privacy，有线等效加密）的连接无线LAN的脚本如下：

```
#!/bin/bash
#文件名: wlan_connect.sh
#用途: 连接无线 LAN

#根据你的设置修改下面的参数
######### PARAMETERS ###########
IFACE=wlan0
IP_ADDR=192.168.1.5
SUBNET_MASK=255.255.255.0
GW=192.168.1.1
HW_ADDR='00:1c:bf:87:25:d2'
#如果不想使用物理地址欺骗，把上面这一行注释掉

ESSID="homenet"
WEP_KEY=8b140b20e7
FREQ=2.462G
#################################
```

7

```
KEY_PART=""

if [[ -n $WEP_KEY ]];
then
  KEY_PART="key $WEP_KEY"
fi
#设置新的配置之前先关闭接口
/sbin/ifconfig $IFACE down

if [ $UID -ne 0 ];
then
  echo "Run as root"
  exit 1;
fi

if [[ -n $HW_ADDR  ]];
then
  /sbin/ifconfig $IFACE hw ether $HW_ADDR
  echo Spoofed MAC ADDRESS to $HW_ADDR
fi

/sbin/iwconfig $IFACE essid $ESSID $KEY_PART freq $FREQ

/sbin/ifconfig $IFACE $IP_ADDR netmask $SUBNET_MASK

route add default gw $GW $IFACE

echo Successfully configured $IFACE
```

7.7.3　工作原理

命令`ifconfig`、`iwconfig`和`route`必须以root用户身份运行，因此在脚本一开始要检查是否为root用户。

无线LAN需要essid、密钥、频率等参数。`essid`是我们想要连接的无线网络的名称。一些网络需要用WEP密钥进行认证，WEP密钥通常是一个5位或10位十六进制数口令。频率则是分配给特定网络的，`iwconfig`命令用它将无线网卡同对应的无线网络联系起来。

我们可以用`iwlist`工具扫描并列出可用的无线网络。用下面的命令进行扫描：

```
# iwlist scan
wlan0     Scan completed :
          Cell 01 - Address: 00:12:17:7B:1C:65
                    Channel:11
                    Frequency:2.462 GHz (Channel 11)
                    Quality=33/70  Signal level=-77 dBm
```

```
Encryption key:on
ESSID:"model-2"
```

可以从扫描结果中的`Frequency:2.462 GHz (Channel 11)`一行中提取频率参数。

> 为简单起见这个示例使用WEP，但需要说明的是，就目前来讲WEP是不安全的。如果你要负责管理无线网络，最好使用Wi-Fi保护访问2（WPA2）加密方式，以确保安全。

7.7.4 参考

1.17节讲解了字符串比较。

7.8 用 SSH 实现无密码自动登录

SSH广泛用于脚本自动化，它使得我们可以在远程主机上执行命令并读取输出。SSH通常使用用户名和密码进行认证，在其执行过程中会提示输入密码。但是在自动化脚本中要求用户输入密码就显然不实际了。因此需要将登录过程自动化。SSH包含了一个内建特性，可以用SSH密钥实现自动登录。这则攻略描述了如何创建SSH密钥并协助实现自动登录。

7.8.1 预备知识

SSH采用了**非对称加密技术**，认证密钥包含两部分：一个公钥和一个私钥。我们可以通过`ssh-keygen`命令创建认证密钥。要想实现自动化认证，公钥必须放置在服务器中（将其加入文件~/.ssh/authorized_keys），与公钥对应的私钥应该放入登录客户机的~/.ssh目录中。另一些与SSH相关的配置信息（例如，arthorized_keys文件的路径与名称）可以通过修改文件/etc/ssh/sshd_config进行配置。

7.8.2 实战演练

设置SSH自动化认证需要两步：

(1) 创建SSH密钥，这用于登录远程主机；

(2) 将生成的公钥传给远程主机，并将其加入文件~/.ssh/authorized_keys中。

输入命令`ssh-keygen`创建SSH密钥，并指定加密算法类型为RSA：

```
$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/slynux/.ssh/id_rsa):
Created directory '/home/slynux/.ssh'.
```

```
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/slynux/.ssh/id_rsa.
Your public key has been saved in /home/slynux/.ssh/id_rsa.pub.
The key fingerprint is:
f7:17:c6:4d:c9:ee:17:00:af:0f:b3:27:a6:9c:0a:05 slynux@slynux-laptop
The key's randomart image is:
+--[ RSA 2048]----+
|           .     |
|            o . .|
|     E       o o.|
|      ...oo |
|       .S .+  +o.|
|      .  . .=....|
|     .+.o...|
|      . . + o.  .|
|       ..+       |
+-----------------+
```

你需要输入一个口令来生成一对公钥和私钥。如果不输入的话，也可以生成密钥，但是这样做可不安全。我们可以编写监控脚本，利用自动登录来登入多台主机。对于这种情况，在运行ssh-keygen命令时，不要填入口令，这样就能够避免在脚本运行时向你索要口令了。

现在 ~/.ssh/id_rsa.pub和 ~/.ssh/id_rsa已经生成了。id_dsa.pub是生成的公钥，id_dsa是生成的私钥。公钥必须添加到远程服务器 ~/.ssh/authorized_keys文件中，这台服务器也正是我们想从当前主机自动登入的那台服务器。

要添加一个密钥文件，可以使用：

```
$ ssh USER@REMOTE_HOST "cat >> ~/.ssh/authorized_keys" < ~/.ssh/id_rsa.pub
Password:
```

在上面的命令中要提供登录密码。

自动登录就已经设置好了。从现在开始，SSH在运行过程中就不会再提示输入密码。你可以用下面的命令来测试：

```
$ ssh USER@REMOTE_HOST uname
Linux
```

这样就不会提示你输入密码了。



多数Linux发布版中有一个叫做ssh-copy-id的工具，它可以自动将私钥加入远程服务器的authorized_keys文件中。用法如下：

```
ssh-copy-id USER@REMOTE_HOST
```

7.9 使用 SSH 进行端口转发

端口转发允许其他计算机利用你的主机来连接到远程服务器上的特定服务。举例而言，假设你的主机分配的IP地址是192.168.1.2，而且还拥有互联网连接。如果你将主机端口8000上的流量转发到www.kernel.org的端口80上，那么其它计算机就可以使用浏览器通过访问http://192.168.1.2:8000来进入Linux内核站点了。来看看如何实现这一切。

7.9.1 实战演练

你可以将本地主机端口上的流量转发到另一台主机上，也可以将远程主机端口上的流量转发到其它主机。按照下面的方法，一旦端口转发设置完毕，你会得到一个shell提示符。在进行转发的过程中，这个shell必须保持打开状态，什么时候想停止端口转发，只需要退出该shell就可以了。

(1) 使用下列命令将本地主机端口8000上的流量转发到www.kernel.org的端口80上：

```
ssh -L 8000:www.kernel.org:80 user@localhost
```

将上述命令中的`user`替换成本地主机上的用户名。

(2) 使用下列命令将远程主机端口8000上的流量转发到www.kernel.org的端口80上：

```
ssh -L 8000:www.kernel.org:80 user@REMOTE_MACHINE
```

将上述命令中的`REMOTE_MACHINE`替换成远程主机的主机名或IP地址，将`user`替换成使用SSH进行访问的用户名。

7.9.2 补充内容

使用非交互模式或者反向端口转发可以使端口转发发挥更大的作用。来看一看这两者的用法。

1. 非交互式端口转发

如果你只是想设置端口转发，而不希望总是保持一个打开状态的shell，那么可以像下面这样使用`ssh`：

```
ssh -fL 8000:www.kernel.org:80 user@localhost -N
```

`-f`指定`ssh`在执行命令前转入后台运行，`-L`指定远程主机的登录名，`-N`告诉`ssh`无需执行命令，只进行端口转发。

2. 反向端口转发

反向端口转发是SSH最强大的特性之一。如果你有一台无法通过互联网进行访问的主机，但

是又希望其他用户可以访问到这台主机上的服务，那就是反向端口转发大显身手的时候了。如果你使用SSH访问一台可以通过互联网访问的远程主机，那么就可以在这台主机上设置反向端口转发，将流量转发到运行服务的本地主机上。

反向端口转发的设置同端口转发非常类似：

```
ssh -R 8000:localhost:80 user@REMOTE_MACHINE
```

上述命令会将远程主机端口8000上的流量转发到本地主机的端口80上。和之前一样，别忘了把REMOTE_MACHINE替换成远程主机的主机名或IP地址。

利用这种方法，如果你在远程主机上浏览http://localhost，那么实际连接的是运行在本地主机端口8000上的Web服务器。

7.10 在本地挂载点上挂载远程驱动器

在执行数据读写操作时，如果可以通过本地挂载点访问远程主机文件系统，那就再好不过了。SSH是网络中最常用的文件传输协议，因此可以利用它和sshfs来实现这一需求。sshfs允许你将远程文件系统挂载到本地挂载点上。让我们看看实现方法。

7.10.1 预备知识

GNU/Linux发布版默认并不包含sshfs。请使用软件包管理器来自行安装。sshfs是FUSE文件系统软件包的一个扩展，允许它支持的操作系统像使用本地文件系统一样挂载各类数据。

有关FUSE更多的信息，请访问http://fuse.sourceforge.net/。

7.10.2 实战演练

将位于远程主机上的文件系统挂载到本地挂载点上：

```
# sshfs -o allow_other user@remotehost:/home/path /mnt/mountpoint
Password:
```

在收到提示时输入用户密码。现在位于远程主机/home/path中的数据就可以通过本地挂载点/mnt/mountpoint来访问了。

完成任务后，可用下面的方法卸载：

```
# umount /mnt/mountpoint
```

7.10.3 参考

7.5节讲解了ssh命令。

7.11 网络流量与端口分析

网络端口是网络应用程序必不可少的参数。应用程序在主机上打开端口，通过远程主机中打开的端口实现远程通信。出于安全方面的考虑，必须留意系统中开放及关闭的端口。恶意软件和Rootkit可能会利用特定的端口及服务运行在系统中，从而使攻击者可以对数据及资源进行未经授权的访问。通过获取开放端口列表以及运行在端口之上的服务，便可以分析并抵御Rootkit，而且这些信息还有助于对其进行清除。开放端口列表不仅能够协助检测恶意软件，而且还能够收集开放端口的相关信息，以便调试网络应用程序。它可以用来分析端口连接及端口侦听功能是否正常。这则攻略讨论了各种用于端口分析的工具。

7.11.1 预备知识

很多命令可用来列出端口以及运行在端口上的服务（例如lsof和netstat）。这些命令都默认包含在所有的GNU/Linux发布版中。

7.11.2 实战演练

要列出系统中的开放端口以及运行在端口上的服务的详细信息，可以使用以下命令：

```
$ lsof -i
COMMAND    PID  USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
firefox-b 2261 slynux   78u  IPv4  63729      0t0  TCP localhost:47797-
>localhost:42486 (ESTABLISHED)
firefox-b 2261 slynux   80u  IPv4  68270      0t0  TCP slynux-laptop.
local:41204->192.168.0.2:3128 (CLOSE_WAIT)
firefox-b 2261 slynux   82u  IPv4  68195      0t0  TCP slynux-laptop.
local:41197->192.168.0.2:3128 (ESTABLISHED)
ssh       3570 slynux    3u  IPv6  30025      0t0  TCP localhost:39263-
>localhost:ssh (ESTABLISHED)
ssh       3836 slynux    3u  IPv4  43431      0t0  TCP slynux-laptop.
local:40414->boneym.mtveurope.org:422 (ESTABLISHED)
GoogleTal 4022 slynux   12u  IPv4  55370      0t0  TCP localhost:42486
(LISTEN)
GoogleTal 4022 slynux   13u  IPv4  55379      0t0  TCP localhost:42486-
>localhost:32955 (ESTABLISHED)
```

lsof的每一项输出都对应着一个打开了特定端口的服务。输出的最后一列类似于：

```
laptop.local:41197->192.168.0.2:3128
```

输出中的laptop.local:41197对应本地主机，192.168.0.2:3128对应远程主机。41197是本地主机当前的开放端口，3128是连接远程主机服务所需要使用的端口。

要列出本地主机当前的开放端口，可以使用：

```
$ lsof -i | grep ":[0-9]\+->" -o | grep "[0-9]\+" -o  | sort | uniq
```

7.11.3　工作原理

第一个grep中使用的正则表达式:[0-9]\+->用来从lsof输出中提取主机端口部分（:34395->）；第二个grep用来提取端口号（数字）。同一个端口可能会有多个连接，因此相同的端口也许会出现多次。为了保证每个端口只显示一次，将端口号排序并打印出不重复的部分。

7.11.4　补充内容

让我们看看另一些用来查看开放端口以及网络流量相关信息的工具。

用netstat查看开放端口与服务

netstat是另一个可用于网络服务分析的命令。讲解netstat的所有特性超出了这则攻略的范围。这里只看如何用它列出服务与端口号。

用netstat -tnp列出开放端口与服务：

```
$ netstat -tnp
(Not all processes could be identified, non-owned process info
will not be shown, you would have to be root to see it all.)
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
PID/Program name
tcp        0      0 192.168.0.82:38163      192.168.0.2:3128
ESTABLISHED 2261/firefox-bin
tcp        0      0 192.168.0.82:38164      192.168.0.2:3128        TIME_
WAIT   -
tcp        0      0 192.168.0.82:40414      193.107.206.24:422
ESTABLISHED 3836/ssh
tcp        0      0 127.0.0.1:42486         127.0.0.1:32955
ESTABLISHED 4022/GoogleTalkPlug
tcp        0      0 192.168.0.82:38152      192.168.0.2:3128
ESTABLISHED 2261/firefox-bin
tcp6       0      0 ::1:22                  ::1:39263
ESTABLISHED -
tcp6       0      0 ::1:39263               ::1:22
ESTABLISHED 3570/ssh
```

7.12　创建套接字

对于已有的应用，例如文件传输、远程shell等，我们有现成的工具可以使用（ftp和ssh）。

但有时候你可能需要一些定制的网络应用。比如写一个脚本，当远程客户连接到主机时执行某些操作。在这则攻略中，我们会创建简单的套接字并用它们进行通信。

7.12.1 预备知识

要实现这一切，需要创建网络套接字，使得我们可以通过TCP/IP网络进行数据传输。最简单的方法就是使用netcat命令（或nc）。我们需要两个套接字：一个用来侦听，一个用来连接。

7.12.2 实战演练

(1) 设置侦听套接字：

```
nc -l 1234
```

这会在本地主机的端口1234上创建一个侦听套接字。

(2) 连接到该套接字：

```
nc HOST 1234
```

如果是在运行着侦听套接字的主机上执行该命令，那么需要将HOST更换成localhost，或将其更换成其他主机的IP地址或主机名。

(3) 要想发送消息，只需要在执行第2步操作的主机终端中输入信息并按回车键就行了。消息会出现在执行第1步操作的主机终端中。

7.12.3 补充内容

网络套接字可不仅能用于发送文本。接着往下看。

在网络上进行快速文件复制

我们可以利用netcat和shell重定向来简化网络中的文件复制操作。

(1) 在接收端执行下列命令：

```
nc -l 1234 > destination_filename
```

(2) 在发送端执行下列命令：

```
nc HOST 1234 < source_filename
```

7.13 互联网连接共享

在如今的计算世界里，我们平时都在使用着各种类型的设备。桌面计算机、上网本、笔记本

电脑、平板电脑、智能耳机，等等。我们需要将这些设备都连入互联网，这通常是通过无线路由器实现的。如果手边没有路由器（或者路由器坏掉了），但又需要共享使用互联网，该怎么办呢？没问题！Linux的`iptables`再加上几个脚本便可助你一臂之力。

7.13.1　预备知识

在这则攻略中，我们使用`iptables`设置**网络地址转换**（Network Address Translation，NAT），使得多个联网设备能够共享互联网连接。你需要使用`iwconfig`命令来获得无线接口的名称。

7.13.2　实战演练

(1) 连接到互联网。在这里我们假设使用的是有线网络连接，通过`eth0`连接到互联网。请按照你个人的实际情况进行修改。

(2) 使用发布版自带的网络管理工具，创建一个adhoc无线连接，设置如下：

- IP地址：10.99.66.55
- 子网掩码：255.255.0.0（16）

(3) 使用下面的shell脚本来实现互联网连接共享：

```
#!/bin/bash
#文件名：netsharing.sh
echo 1 > /proc/sys/net/ipv4/ip_forward
iptables -A FORWARD -i $1 -o $2 -s 10.99.0.0/16 -m conntrack
--ctstate NEW -j ACCEPT
iptables -A FORWARD -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
iptables -A POSTROUTING -t nat -j MASQUERADE
```

(4) 执行脚本：

```
./netsharing.sh eth0 wlan0
```

其中，`eth0`是连接到互联网的接口，`wlan0`是用于共享互联网连接的无线接口。

(5) 将设备连接到刚才创建的无线网络：

- IP地址：10.99.66.56（以此类推）。
- 子网掩码：255.255.0.0。

> 要想更为便捷，可以在主机上安装DHCP和DNS服务器，这样就不必手动配置IP地址了。你可以使用一个叫做`dnsmasq`的工具来进行DHCP和DNS操作。

7.14 使用 iptables 架设简易防火墙

防火墙是一种网络服务，它可以用来过滤、阻塞不需要的网络流量，允许正常的网络流量通过。Linux中最为强大的工具非iptables莫属，它目前已经被集成到了内核中。

7.14.1 实战演练

如今所有的Linux发布版中默认都包含了iptables。我们接下来会看到一些典型的iptables的配置方法。

(1) 阻塞发送到特定IP地址的流量：

```
#iptables -A OUTPUT -d 8.8.8.8 -j DROP
```

如果在执行iptables命令之前，从另一个终端中运行PING 8.8.8.8，你会看到如下内容：

```
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_req=1 ttl=56 time=221 ms
64 bytes from 8.8.8.8: icmp_req=2 ttl=56 time=221 ms
ping: sendmsg: Operation not permitted
ping: sendmsg: Operation not permitted
```

ping命令在执行到第三次的时候失败了，这是因为我们使用iptables将所有发送到8.8.8.8的流量给丢弃了。

(2) 阻塞发送到特定端口的流量：

```
#iptables -A OUTPUT -p tcp -dport 21 -j DROP
$ ftp ftp.kde.org
ftp: connect: Connection timed out
```


7.14.2 工作原理

iptables是Linux系统中用来假设防火墙的标准命令。iptables中的第一个选项-A表明向链（chain）中添加一条新的规则，该规则由后续参数给出。链就是一组规则的集合，在这则攻略中我们使用的是OUTPUT链，它可以对所有出站（outgoing）的流量进行控制。

在第一个例子中，-d指定了所要匹配的分组目的地址。随后使用-j来使iptables丢弃（DROP）符合条件的分组。

在第二个例子中，-p指定该规则是适用于TCP，-dport指定了对应的端口。这样我们就可以阻塞所有出站的FTP流量了。

7.14.3　补充内容

在使用iptables命令时，你可能希望清除对iptables链所做出的所有改动。可以使用下面的命令：

```
#iptables --flush
```

第 8 章

当个好管家

本章内容

- 统计磁盘使用情况
- 计算命令的执行时间
- 登录用户、启动日志及启动故障的相关信息
- 列出1小时内占用CPU最多的10个进程
- 用watch监视命令输出
- 记录文件及目录访问
- 用logrotate管理日志文件
- 用syslog记录日志
- 通过监视用户登录找出入侵者
- 监视远程磁盘的健康情况
- 找出系统中用户的活动时段
- 电源使用的测量与优化
- 监视磁盘活动
- 检查磁盘及文件系统错误

8.1 简介

操作系统是由一系列不同用途的系统软件组成的。为了了解这些软件是否工作正常，最好能够对其进行监视。我们可以使用一项被称为日志记录（logging）的技术，借助这项技术，应用程序在运行的时候，会将重要的信息写入某个文件中。该文件的内容能够用来了解特定程序或守护进程的操作过程。如果应用程序或服务发生了崩溃，这些信息有助于我们进行调试和故障修复。

在本章中，我们要和监视系统活动的各种命令打交道，同时还要学习日志技术及其使用方法。

8.2 监视磁盘使用情况

磁盘空间是一种有限资源。我们经常要统计存储介质（例如硬盘）的使用情况，以便确定可用空间。当可用空间开始捉襟见肘时，我们就得找到大体积的文件，删除或移走它们，以便腾出空间。除此之外，磁盘使用管理也用于shell脚本环境。这则攻略将会讲解用于磁盘管理的各类命令及其选项。

8.2.1　预备知识

df和du是Linux中用于统计磁盘使用情况的两个重要命令。df是disk free的缩写，du是disk usage的缩写。让我们看看如何用它们执行各种涉及磁盘使用情况的任务。

8.2.2　实战演练

找出某个文件（或多个文件）占用的磁盘空间：

```
$ du  FILENAME1 FILENAME2 ..
```

例如：

```
$ du file.txt
4
```

> 统计结果默认以字节作为计量单位。

要获得某个目录中所有文件的磁盘使用情况，并在每一行中显示各个文件的磁盘占用详情，可以使用：

```
$ du -a DIRECTORY
```

-a递归地输出指定目录或多个目录中所有文件的统计结果。

> 执行du DIRECTORY也可以输出类似的结果，但是它只会显示子目录使用的磁盘空间，而不显示每个文件占用空间的情况。要想显示文件的磁盘使用情况，必须使用 -a。

例如：

```
$  du -a test
4  test/output.txt
4  test/process_log.sh
4  test/pcpu.sh
16  test
```

使用du DIRECTORY的输出如下：

```
$ du test
16  test
```

8.2.3　补充内容

让我们看看du命令的其他用法。

1. 以KB、MB或块（block）为单位显示磁盘使用情况

命令du默认显示文件占用的总字节数，但是以标准单位KB、MB或GB显示磁盘使用情况更方便人们阅读。要采用这种更友好的格式进行打印，可以使用选项 -h：

```
du -h FILENAME
```

例如：

```
$ du -h test/pcpu.sh
4.0K  test/pcpu.sh
# 可以接受多个文件参数
```

或者

```
# du -h DIRECTORY
$ du -h hack/
16K  hack/
```

2. 显示磁盘使用总计

假如我们需要计算所有文件或目录总共占用了多少磁盘空间，那么显示单个文件的使用情况显然没什么用。du的选项 -c可以输出作为命令参数的所有文件和目录的磁盘使用情况，它会在输出结果末尾加上一行总计。语法如下：

```
$ du -c FILENAME1 FILENAME2..
```

例如：

```
du -c process_log.shpcpu.sh
4  process_log.sh
4  pcpu.sh
8  total
```

或者

```
$ du  -c DIRECTORY
```

例如：

```
$ du -c test/
16  test/
16  total
```

或者

```
$ du -c *.txt
# 通配符
```

-c可以同 -a、-h等选项组合使用。如果不使用-c，也可以得到同样的输出。唯一不同的是选项-c会添加一行磁盘使用情况总计。

另一个选项 -s（summarize，合计）则只输出合计数据。它可以配合 -h打印出人们易读的格式。这个命令在实践中经常使用，其语法如下：

```
$ du -s FILES(s)
$ du -sh DIRECTORY
```

例如：

```
$ du -sh slynux
680K  slynux
```

3. 用特定的单位打印文件

我们可以强制du使用特定的单位打印磁盘使用情况。举例如下。

- 打印以字节（默认输出）为单位的文件大小：

```
$ du -b FILE(s)
```

- 打印以KB为单位的文件大小：

```
$ du -k FILE(s)
```

- 打印以MB为单位的文件大小：

```
$ du -m FILE(s)
```

- 打印以指定块为单位的文件大小：

```
$ du -B BLOCK_SIZE FILE(s)
```

其中BLOCK_SIZE以字节为单位。

下面的例子中包含了这些命令：

```
$ du pcpu.sh
4  pcpu.sh
$ du -b pcpu.sh
439   pcpu.sh
$ du -k pcpu.sh
4  pcpu.sh
$ du -m pcpu.sh
1  pcpu.sh
$ du -B 4  pcpu.sh
1024  pcpu.sh
```

4. 从磁盘使用统计中排除部分文件

有时候我们需要从磁盘使用统计中排除部分文件。可以使用两种方法指定需要排除的文件：

(1) 通配符

按照下面的方法指定一个通配符：

```
$ du --exclude "WILDCARD" DIRECTORY
```

例如：

```
$ du --exclude "*.txt" FILES(s)
# 排除所有的.txt文件
```

(2) 排除列表

从文件中获取需要排除的文件列表：

```
$ du --exclude-from EXCLUDE.txt DIRECTORY
# EXCLUDE.txt包含了需要排除的文件列表
```

还有其他的选项可以很方便地用来限制磁盘使用情况统计的范围。我们可以用--max-depth指定du应该遍历的目录层次的最大深度。将深度指定为1，可以统计当前目录下的所有文件占用内存的情况。将深度指定为2，可以统计当前目录以及下一级子目录中文件占用内存的情况，但不包括第二级子目录。

例如：

```
$ du --max-depth 2 DIRECTORY
```

当使用du时，要确保对其遍历的目录和文件拥有适合的读权限。

> 可以用选项-x来限制du，使它只能够对单个文件系统进行遍历。假设运行du DIRECTORY，它会递归地遍历每一个可能存在的子目录。目录层次中的某个子目录可能就是一个挂载点（例如，/mnt/sda1是/mnt的子目录，同时也是设备/dev/sda1的挂载点）。du会遍历挂载点并统计设备文件系统的磁盘使用情况。为了避免du的这种行为，可以在使用du其他选项的同时加上选项 -x。du -x / 会将/mnt/中的所有挂载点排除在磁盘使用统计之外。

5. 找出指定目录中最大的10个文件

我们经常会碰上找出大体积文件的活儿，一般是要将这些大文件删除或移走。可以用du和sort轻松地完成这项任务：

```
$ du -ak SOURCE_DIR | sort -nrk 1 | head
```

其中，-a使得du遍历SOURCE_DIR并计算其中所有文件的大小。由于指定了选项-k，输出的第一列会包含以KB为单位的文件大小，第二列则会包含文件或文件夹的名称。

sort对第一列依数值逆序排序。head用来显示前10行输出。例如：

```
$ du -ak /home/slynux | sort -nrk 1 | head -n 4
50220 /home/slynux
43296 /home/slynux/.mozilla
43284 /home/slynux/.mozilla/firefox
43276 /home/slynux/.mozilla/firefox/8c22khxc.default
```

上面这个单行脚本的缺点在于它在结果中包含了目录。如果我们只是需要找出最大的文件而不是目录，可以按照下面的方法改进脚本，使它只输出最大的文件：

```
$ find . -type f -exec du -k {} \; | sort -nrk 1 | head
```

利用find替du将文件过滤出来，而无需使用du进行递归遍历。

6. 磁盘可用空间信息

du提供磁盘使用情况信息，而df提供磁盘可用空间信息。df的-h选项会以易读的格式打印磁盘空间信息。例如：

```
$ df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/sda1             9.2G  2.2G  6.6G  25% /
none                  497M  240K  497M   1% /dev
none                  502M  168K  501M   1% /dev/shm
none                  502M   88K  501M   1% /var/run
none                  502M     0  502M   0% /var/lock
none                  502M     0  502M   0% /lib/init/rw
none                  9.2G  2.2G  6.6G  25% /var/lib/ureadahead/debugfs
```

8.3　计算命令执行时间

当测试一个应用程序的效率或比较不同的算法时，其执行时间非常重要。好算法的执行时间应该很短。让我们看看如何来计算执行时间。

8.3.1　实战演练

(1) 要测试执行时间，只需要将time放在想要运行的命令之前。例如：

```
$ time COMMAND
```

命令COMMAND会执行并生成输出。除此之外，time命令会将命令的执行时间添加到stderr中。例如：

```
$ time ls
test.txt
next.txt
real    0m0.008s
user    0m0.001s
sys     0m0.003s
```

输出中分别显示了执行该命令所花费的real时间、user时间以及sys时间。

> time命令的可执行二进制文件位于/usr/bin/time，还有一个shell内建命令也叫做time。当运行time时，默认调用的是shell的内建命令。shell内建的time命令选项有限。因此，如果我们需要使用另外的功能，就应该使用可执行文件time的绝对路径（/usr/bin/time）。

(2) 可以用选项-o filename将相关的时间统计信息写入文件：

```
$ /usr/bin/time -o output.txt COMMAND
```

文件名应该出现在选项-o之后。

要将命令执行时间添加到文件而不影响其原有内容，使用选项-a以及-o：

```
$ /usr/bin/time -a -o output.txt COMMAND
```

(3) 我们也可以使用选项-f，利用格式字符串来格式化时间输出。格式字符串由对应于特定选项的参数组成，这些参数以 % 作为前缀。real时间、user 时间、sys 时间的格式字符串分别如下：

- real: %e
- user: %U
- sys: %S

通过结合参数字符串，我们就可以创建格式化输出：

```
$ /usr/bin/time -f "FORMAT STRING" COMMAND
```


例如：

```
$ /usr/bin/time -f "Time: %U" -a -o timing.log uname
Linux
```

其中，%U是user时间的参数。

格式化输出生成后被写入标准输出，命令的执行时间信息被写入标准错误。我们可以用重定向操作符（>）对格式化输出重定向，用错误重定向操作符（2>）对时间信息重定向。

例如：

```
$ /usr/bin/time -f "Time: %U" uname> command_output.txt 2>time.log
$ cat time.log
Time: 0.00
$ cat command_output.txt
Linux
```

(4) 使用参数%Z显示系统页面大小：

```
$ /usr/bin/time -f "Page size: %Z bytes" ls> /dev/null
Page size: 4096 bytes
```

这里并不需要被计时命令的输出（即ls），因此为了使其不显示在终端中，我们将标准输出重定向到了/dev/null设备。

还有很多可用格式字符串，请阅读man time以获取更多的细节信息。

8.3.2　工作原理

有以下三种不同类型的时间。

- Real时间指的是挂钟时间（wall clock time），也就是命令从开始执行到结束的时间。这段时间包括其他进程所占用的时间片（time slice）以及进程被阻塞时所花费的时间（例如，为等待I/O操作完成所用的时间）。
- User时间是指进程花费在用户模式（内核之外）中的CPU时间。这是唯一真正用于执行进程所花费的时间。执行其他进程以及花费在阻塞状态中的时间并没有计算在内。
- Sys时间是指进程花费在内核中的CPU时间。它代表在内核中执行系统调用所使用的时间，这和库代码（library code）不同，后者仍旧运行在用户空间。与“user时间”类似，这也是真正由进程使用的CPU时间。

通过time命令可以获得进程的很多细节信息，包括退出状态、接收到的信号数量、进程上下文的切换次数等。每一个参数都可以用适合的格式字符串显示。

表8-1展示了一些可以使用的参数。

表　8-1

参　数	描　述
%C	进行计时的命令名称以及命令行参数
%D	进程非共享数据区域的大小，以KB为单位
%E	进程使用的real时间（挂钟时间），显示格式为[小时:]分钟:秒
%x	命令的退出状态
%k	进程接收到的信号数量
%W	进程被交换出主存的次数
%Z	系统的页面大小。这是一个系统常量，但在不同的系统中，这个常量值也不同
%P	进程所获得的CPU时间百分比。这个值等于user+system时间除以总运行时间。结果以百分比形式显示
%K	进程的平均总（data+stack+text）内存使用量，以KB为单位
%w	进程主动进行上下文切换的次数，例如等待I/O操作完成
%c	进程被迫进行上下文切换的次数（由于时间片到期）

8.4　收集与当前登录用户、启动日志及启动故障的相关信息

收集与操作环境、当前登录用户、主机加电时间、启动故障等相关的信息很有用处。在这则攻略会讲解一些用于收集活动主机信息的相关命令。

8.4.1　预备知识

接下来将介绍以下命令：`who`、`w`、`users`、`uptime`、`last`和`lastb`。

8.4.2　实战演练

(1) 获取当前登录用户的相关信息：

```
$ who
slynux   pts/0   2010-09-29 05:24 (slynuxs-macbook-pro.local)
slynux   tty7    2010-09-29 07:08 (:0)
```

该命令会显示出登录名、用户所使用的TTY、登录时间以及登录用户的远程主机名（或者X显示信息）。

> TTY（该术语取自TeleTYperwriter）是与文本终端相关联的设备文件。当用户生成一个新终端时，对应的设备文件就会出现在/dev/中（例如 /dev/pts/3）。可以通过输入并执行命令`tty`来获得当前终端的设备路径。

(2) 获得有关登录用户更详细的信息：

```
$ w
 07:09:05 up  1:45,  2 users,  load average: 0.12, 0.06, 0.02
USER     TTY      FROM     LOGIN@   IDLE  JCPU PCPU WHAT
slynux   pts/0    slynuxs 05:24   0.00s  0.65s 0.11s sshd: slynux
slynux   tty7     :0       07:08   1:45m  3.28s 0.26s gnome-session
```

第一行列出了当前时间，系统运行时间，当前登录的用户数量以及过去的1分钟、5分钟、15分钟内的系统平均负载。接下来的每一行显示了每一个登录用户的详细信息，其中包括登录名、TTY、远程主机、登录时间、空闲时间、自该用户登录后所使用的总CPU时间、当前运行进程所使用的CPU时间以及进程所对应的命令行。

> uptime命令输出中的平均负载（load average）是表明系统负载量的一个参数。在第9章我们会对此进行详细地解释。

(3) 列出当前登录主机的用户列表：

```
$ users
slynux slynux slynux hacker
```

如果某个用户打开了多个伪终端，那么该用户会被多次显示。在上面的输出中，用户slynux打开了3个伪终端。排除重复用户出现的最简单的方法是使用sort和uniq进行过滤：

```
$ users | tr ' ' '\n' | sort | uniq
slynux
hacker
```

利用tr将' '替换成'\n'，然后用sort和uniq为每个用户生成唯一的输出。

(4) 查看系统已经加电运行了多长时间：

```
$ uptime
 21:44:33 up  3:17,  8 users,  load average: 0.09, 0.14, 0.09
```

单词up之后的时间表明了系统已经加电运行了多久。我们可以编写一个简单的单行脚本来提取运行时间：

```
$ uptime | grep -Po '\d{2}\:\d{2}\:\d{2}'
```

这条命令利用grep和perl风格的正则表达式来提取由冒号分隔的3组两位数字。

(5) 获取上一次启动以及用户登录会话的信息：

```
$ last
slynux   tty7         :0              Tue Sep 28 18:27   still logged in
```

```
reboot   system boot  2.6.32-21-generic Tue Sep 28 18:10 - 21:46  (03:35)
slynux   pts/0        :0.0              Tue Sep 28 05:31 - crash  (12:39)
```

last命令可以提供登录会话信息。它实际上是一个系统登录日志，包括了登录tty、登录时间、状态等信息。

last命令以日志文件/var/log/wtmp作为输入日志数据。它也可以用选项-f明确地指定日志文件。例如：

```
$ last -f /var/log/wtmp
```

(6) 获取单个用户登录会话的信息：

```
$ last USER
```

(7) 获取重启会话（reboot session）信息：

```
$ last reboot
reboot   system boot  2.6.32-21-generi Tue Sep 28 18:10 - 21:48  (03:37)
reboot   system boot  2.6.32-21-generi Tue Sep 28 05:14 - 21:48  (16:33)
```

(8) 获取失败的用户登录会话信息：

```
# lastb
test     tty8         :0               Wed Dec 15 03:56 - 03:56  (00:00)
slynux   tty8         :0               Wed Dec 15 03:55 - 03:55  (00:00)
```

你必须以root用户的身份运行lastb。

8.5　列出 1 小时内占用 CPU 最多的 10 个进程

CPU是一种重要的资源，最好能够跟踪某个阶段内占用CPU最多的进程。通过监视一段时间内CPU的使用情况，我们可以找出长期占用CPU的进程并对其进行优化，提高CPU使用效率。我们在这则攻略中讨论了进程监视与日志记录。

8.5.1　预备知识

ps命令用于收集系统中进程的详细信息。这些信息包括CPU使用情况、正在执行的命令、内存占用、进程状态等。记录在一小时内的CPU占用情况，然后灵活地运用ps以及文本处理就可以找出占用CPU最多的10个进程。关于ps命令的更多细节，请参考第9章。

8.5.2　实战演练

让我们看看用于监视并计算一小时内CPU使用情况的shell脚本：

```
#!/bin/bash
#文件名: pcpu_usage.sh
#用途:计算1个小时内进程的CPU占用情况

SECS=3600
UNIT_TIME=60

#将SECS更改成需要进行监视的总秒数
#UNIT_TIME是取样的时间间隔，单位是秒

STEPS=$(( $SECS / $UNIT_TIME ))

echo Watching CPU usage... ;

for((i=0;i<STEPS;i++))
do
  ps -eocomm,pcpu | tail -n +2 >> /tmp/cpu_usage.$$
  sleep $UNIT_TIME
done

echo
echo CPU eaters :

cat /tmp/cpu_usage.$$ | \
awk '
{ process[$1]+=$2; }
END{
  for(i in process)
  {
    printf("%-20s %s\n",i, process[i]) ;
  }

   }' | sort -nrk 2 | head

rm /tmp/cpu_usage.$$
#删除临时日志文件
```

输出如下:

```
$ ./pcpu_usage.sh
Watching CPU usage...
CPU eaters :
Xorg        20
firefox-bin   15
bash        3
evince      2
pulseaudio    1.0
pcpu.sh         0.3
wpa_supplicant  0
```

```
wnck-applet      0
watchdog/0       0
usb-storage      0
```

8.5.3 工作原理

在上面的脚本中，主要的输入源是ps -eo comm,pcpu，其中comm表示命令名（command name），pcpu表示CPU使用率（CUP usage in percent）。该命令输出所有进程名及CPU使用率。每个进程对应一行输出。因为需要监视一小时内CPU的使用情况，所以我们得在一个每次迭代时间为60秒的for循环中不停地用ps -eo comm,pcpu | tail -n +2来获取CPU的使用统计数据，并将这些数据添加到文件 /tmp/cpu_usage.$$ 中。60秒的迭代时间通过sleep 60来提供。这就使得每分钟执行一次ps。tail -n +2用来将 ps 输出中的头部和 COMMAND %CPU剔除。

cpu_usage.$$ 中的 $$ 表示当前脚本的进程 ID。假设进程 ID为1345，那么在脚本执行时它会被替换成 /tmp/cpu_usage.1345。因为这是一个临时文件，所以我们把它放在 /tmp中。

统计文件在1小时后就准备妥当了，文件中包含了60项，分别对应每分钟的进程状态。然后用awk求出每个进程总的CPU使用情况。我们用了一个关联数组来统计CPU使用情况。其中进程名作为数组索引。最后根据总的CPU使用情况依数值逆序排序，并通过head获得前10项。

8.5.4 参考

- 4.6节讲解了awk命令。
- 3.13节讲解了tail命令。

8.6 使用 **watch** 监视命令输出

我们可能需要在一段时期内以固定的间隔时间不断监视某个命令的输出。例如在复制大文件时，我们希望看到不断增长的文件大小。要做到这一点，我们可以使用watch命令来执行du命令，不断地重复输出。这则攻略会告诉你实现方法。

8.6.1 实战演练

watch命令可以用来在终端中以固定的间隔监视命令输出。该命令语法如下：

```
$ watch COMMAND
```

例如：

```
$ watch ls
```

或者

```
$ watch 'COMMANDS'
```

例如：

```
$ watch 'ls -l | grep "^d"'
# 只列出目录
```

命令默认每2秒更新一次输出。

我们可以用 -n SECOND指定更新输出的时间间隔。例如：

```
$ watch -n 5 'ls -l'
#以5秒为间隔，监视ls -l的输出
```

8.6.2　补充内容

让我们再来看一下watch命令的其他特性。

突出标示watch输出中的差异

watch有一个选项可以将时间间隔前后的命令输出的差异之处以不同颜色突出标示出来。选项 -d可以启用这一功能：

```
$ watch -d 'COMMANDS'
```

8.7　记录文件及目录访问

记录文件及目录访问对于跟踪文件和目录的变化很有帮助。这则攻略会讲解如何记录这种类型的访问。

8.7.1　预备知识

inotifywait命令可以用来收集有关文件访问的信息。Linux发布版默认并没有包含这个命令，你得用软件包管理器自行安装inotify-tools。这个命令还需要将inotify支持编译入Linux内核，好在大多数新的GNU/Linux发布版已经都在内核中启用了inotify。

8.7.2　实战演练

来看看用来监视目录访问的shell脚本：

```
#/bin/bash
#文件名：watchdir.sh
```

```
#用途:监视目录访问
path=$1
#将目录或文件路径作为脚本参数

inotifywait -m -r -e create,move,delete $path  -q
```

输出样例如下：

```
$ ./watchdir.sh .
./ CREATE new
./ MOVED_FROM new
./ MOVED_TO news
./ DELETE news
```

8.7.3 工作原理

上面的脚本记录给定了路径中文件或目录的创建、移动以及删除。选项 -m表明要持续监视变化，而不是在事件发生之后退出。-r允许采用递归形式监视目录（忽略符号链接）。-e 指定需要监视的事件列表。-q 用于减少冗余信息，只打印出所需要的信息。命令输出可以被重定向到日志文件。

我们可以从事件列表中添加或删除事件。一些重要的事件如表8-2所示。

表 8-2

事件	描述
访问（access）	读取文件
修改（modify）	文件内容被修改
属性（attrib）	文件元数据被修改
移动（move）	移动文件操作
创建（create）	生成新文件
打开（open）	打开文件操作
关闭（close）	关闭文件操作
删除（delete）	文件被删除

8.8 用 logrotate 管理日志文件

日志文件是Linux系统维护中必不可少的组成部分，它可以帮助跟踪系统内各种服务中出现的事件。这既有助于排查问题，也可以为活动主机提供统计数据。随着时间的推移，日志文件会变得越来越大，因而对于日志文件的管理就必不可少了。我们将用一种被称为**轮替**（rotation）的技术来限制日志文件的体积，一旦它超过了限定的大小，就对其内容进行抽取（strip），同时将

日志文件中的旧条目存储到日志目录中的归档文件内。旧的日志文件就会得以保存以便随后参阅。让我们看看如何对日志文件进行轮替及存储。

8.8.1　预备知识

logrotate是每一位Linux系统管理员都应该了解的命令。它能够将日志文件的大小限制在给定的SIZE内。日志记录程序会将信息添加到日志文件中。最新添加的信息总是出现在日志文件的尾部。logrotate根据配置文件扫描特定的日志文件。它在保留日志文件中最新添加的100KB内容（假设指定SIZE = 100k）的同时，将剩下的数据（较旧的日志数据）移入新文件logfile_name.1。随着该日志文件（logfile_name.1）中的内容越来越多，逐渐超出了SIZE规定的定额，logrotate就会再用最近的内容更新日志文件，然后用较旧的内容创建logfile_name.2。整个过程可以轻松地使用logrotate进行配置。logrotate还可以将旧的日志文件压缩成logfile_name.1. gz、logfile_name.2.gz等。是否选择压缩旧日志文件也可以通过logrotate来配置。

8.8.2　实战演练

logrotate的配置目录位于/etc/logrotate.d。如果列出这个目录中的内容，你会发现很多其他的日志文件配置。

我们可以为自己的日志文件（比如/var/log/program.log）编写一个特定的配置：

```
$ cat /etc/logrotate.d/program
/var/log/program.log {
missingok
notifempty
size 30k
  compress
weekly
  rotate 5
create 0600 root root
}
```

这就是全部的内容。其中，/var/log/program.log指定了日志文件路径。旧的日志文件归档之后也放入同一个目录中。

8.8.3　工作原理

表8.3列出了配置文件中各个参数的含义。

表 8-3

参 数	描 述
`missingok`	如果日志文件丢失，则忽略；然后返回（不对日志文件进行轮替）
`notifempty`	仅当源日志文件非空时才对其进行轮替
`size 30k`	限制实施轮替的日志文件的大小。可以用1M表示1MB
`compress`	允许用`gzip`压缩较旧的日志
`weekly`	指定进行轮替的时间间隔。可以是`weekly`、`yearly`或`daily`
`rotate 5`	这是需要保留的旧日志文件的归档数量。在这里指定的是5，所以这些文件名将会是program.log.1.gz、program.log.2.gz等直到program.log.5.gz
`create 0600 root root`	指定所要创建的归档文件的模式、用户以及用户组

表8-3中的选项都是可选的。我们只在`logrotat`配置文件中指定所需的选项即可。如果想了解更多的可用选项，请参考`logrotate`的手册页（http://linux.die.net/man/8/logrotate）。

8.9 用 syslog 记录日志

通常与各种守护进程和应用程序相关的日志文件都会放在/var/log目录中，因为这是存储日志文件的公共目录。如果你读过日志文件，就会看出它们都采用了一种通用的格式。在Linux系统中，由守护进程`sylogd`使用`syslog`协议负责在 /var/log中创建并写入日志信息。每一个标准应用进程都可以利用`syslog`记录日志信息。在这则攻略中，我们将讨论如何在脚本中用`syslogd`进行日志信息的记录。

8.9.1 预备知识

日志文件有助于我们推断系统出现了什么故障。在编写重要的应用程序时，应当将应用程序的执行过程记录在日志文件中。我们接下来将学习使用命令`logger`通过`syslogd`记录日志。在学习如何往日志文件中写入信息之前，先看看Linux中一些重要的日志文件，如表8-4所示。

表 8-4

日志文件	描 述
`/var/log/boot.log`	系统启动信息
`/var/log/httpd`	Apache Web服务器日志
`/var/log/messages`	发布内核启动信息
`/var/log/auth.log`	用户认证日志
`/var/log/dmesg`	系统启动信息
`/var/log/mail.log`	邮件服务器日志
`/var/log/Xorg.0.log`	X服务器日志

8.9.2　实战演练

下面来看看如何使用logger创建及管理日志信息。

(1) 向系统日志文件/var/log/message中写入日志信息：

```
$ logger LOG_MESSAGE
```

例如：

```
$ logger This is a test log line

$ tail -n 1 /var/log/messages
Sep 29 07:47:44 slynux-laptop slynux: This is a test log line
```

/var/log/messages是一个通用日志文件。如果使用logger命令，它默认将日志信息记录到 /var/log/messages中。

(2) 如果要记录特定的标记（tag），可以使用：

```
$ logger -t TAG This is a message

$ tail -n 1 /var/log/messages
Sep 29 07:48:42 slynux-laptop TAG: This is a message
```

Syslog处理/var/log下的多个日志文件。但是当logger发送消息时，它用标记字符串来确定应该记录到哪一个日志文件中。syslogd使用与日志相关联的TAG来决定应该将其记录到哪一个文件中。你可以从/etc/rsyslog.d/目录下的配置文件中看到标记字符串以及与其相关联的日志文件。

(3) 要将另一个日志文件的最后一行记录到系统日志中，可以使用：

```
$ logger -f /var/log/source.log
```

8.9.3　参考

3.13节讲解了head和tail命令。

8.10　通过监视用户登录找出入侵者

日志文件可以用于收集系统状态的详细有关信息。下面是一个脚本相关的问题描述。

我们有一个通过SSH连接到Internet的系统。很多攻击者试图登入这个系统，因此需要利用shell脚本来设计一个入侵检测系统。入侵者定义为：屡次试图登入系统达两分钟以上，并且期间的登录过程全部失败。凡是这类用户都应该被检测出来并生成包含以下细节信息的报告：

- ❑ 试图登录的账户；
- ❑ 试图登录的次数；
- ❑ 攻击者的IP地址；
- ❑ IP地址所对应的主机；
- ❑ 进行登录的时间段。

8.10.1 预备知识

我们可以编写一个shell脚本，对日志文件进行扫描并从中收集所需要的信息。为了处理SSH登录失败的情况，还得知道用户认证会话日志会被记录在日志文件/var/log/auth.log中。脚本需要扫描这个日志文件来检测出失败的登录信息，执行各种检查来获取所需要的数据。我们可以用`host`命令找出IP地址所对应的主机。

8.10.2 实战演练

接下来编写一个入侵检测脚本，它利用用户认证日志文件来生成有关入侵者的报告：

```
#!/bin/bash
#文件名:intruder_detect.sh
#用途:入侵报告工具，以auth.log作为日志文件
AUTHLOG=/var/log/auth.log

if [[ -n $1 ]];
then
  AUTHLOG=$1
  echo Using Log file : $AUTHLOG
fi

LOG=/tmp/valid.$$.log
grep -v "invalid" $AUTHLOG > $LOG
users=$(grep "Failed password" $LOG | awk '{ print $(NF-5) }' | sort | uniq)


printf "%-5s|%-10s|%-10s|%-13s|%-33s|%s\n" "Sr#" "User" "Attempts" "IP address"
"Host_Mapping" "Time range"


ucount=0;

ip_list="$(egrep -o "[0-9]+\.[0-9]+\.[0-9]+\.[0-9]+" $LOG | sort | uniq)"

for ip in $ip_list;
do
  grep $ip $LOG > /tmp/temp.$$.log
```

8

```
  for user in $users;
  do
    grep $user /tmp/temp.$$.log> /tmp/$$.log
    cut -c-16 /tmp/$$.log > $$.time
    tstart=$(head -1 $$.time);
    start=$(date -d "$tstart" "+%s");
    tend=$(tail -1 $$.time);
    end=$(date -d "$tend" "+%s")

    limit=$(( $end - $start ))

    if [ $limit -gt 120 ];
    then
      let ucount++;


      IP=$(egrep -o "[0-9]+\.[0-9]+\.[0-9]+\.[0-9]+" /tmp/$$.log | head -1 );

      TIME_RANGE="$tstart-->$tend"

      ATTEMPTS=$(cat /tmp/$$.log|wc -l);

      HOST=$(host $IP | awk '{ print $NF }' )


    printf "%-5s|%-10s|%-10s|%-10s|%-33s|%-s\n" "$ucount" "$user" "$ATTEMPTS" "$IP"
"$HOST" "$TIME_RANGE";
    fi
  done
done


rm/tmp/valid.$$.log/tmp/$$.log $$.time/tmp/temp.$$.log 2>/dev/null
```

输出样例如下：

```
slynux@slynux-laptop:~$ ./intruder_detect.sh sampleauth.log
Using Log file : sampleauth.log

Sr#   |User    |Attempts|IP address    |Host_Mapping |Time range
1     |alice   |3       |203.110.250.34|attk1.foo.com|Oct 29 05:28:59 -->Oct 29 05:31:59
2     |bob1    |3       |203.110.251.31|attk2.foo.com|Oct 29 05:21:52 -->Oct 29 05:29:52
3     |bob2    |3       |203.110.250.34|attk1.foo.com|Oct 29 05:22:59 -->Oct 29 05:25:52
4     |gvraju  |20      |203.110.251.31|attk2.foo.com|Oct 28 04:37:10 -->Oct 29 05:19:09
5     |root    |21      |203.110.253.32|attk3.foo.com|Oct 29 05:18:01 -->Oct 29 05:37:01
```

8.10.3　工作原理

在intruder_detect.sh脚本中，我们将auth.log文件作为输入。也可以用脚本的命令行参数来提

供一个日志文件作为输入，或者默认读取/var/log/auth.log。我们只需要记录合法用户的登录日志详情。如果有非法用户企图登录，则类似于`Failed password for invalid user bob from 203.83. 248.32 port 7016 ssh2`的日志就会出现在auth.log中。因此我们需要排除日志文件中所有包含`invalid`的行。`grep`命令的反转选项（`-v`）可以用来移除对应非法用户的所有日志内容。下一步是找出试图登录并且失败的用户列表。对于密码错误，SSH会记录类似的日志信息：`sshd[21197]: Failed password for bob1 from 203.83.248.32 port 50035 ssh2`，所以我们应找出所有包含"`failed password`"的行。

接下来，要找出所有不重复的IP地址以提取对应于每一个IP地址的日志行。提取IP地址列表可以用匹配IP地址的正则表达式和`egrep`命令来完成。用`for`循环对IP地址进行迭代，并用`grep`找出对应的日志行并将其写入临时文件。日志行中倒数第6个单词是用户名（例如bob1），可用`awk`命令提取用户名（倒数第6个单词）。`NF`返回最后一个单词的列号，因此`NF -5`就是倒数第6个单词的列号。我们再用`sort`和`uniq`生成一个没有重复的用户列表。

现在我们要收集表明登录失败的日志行，这些行中包含了用户名。`for`循环用来读取对应每一位用户的日志行，并将这些行写入临时文件。每一行的前16个字符是时间戳，可用`cut`命令进行提取。一旦得到了一个用户所有登录失败的时间戳，就要检查第一次和最后一次试图登录之间的时间差。第一行日志对应第一次登录，最后一行日志对应最后一次登录。我们用`head -1`提取首行，用`tail -1`提取末行。这样就有了首次登录（`tstart`）和末次登录（`tends`）的字符串格式的时间戳。使用`date`命令，我们可以将字符串形式的日期转换成Unix纪元时的总秒数（1.10节讲解了Unix纪元时）。

变量`start`和`end`中包含着秒数，分别对应于字符串形式的起始时间戳。对这两个时间求差，并检查差值是否大于2分钟（即120秒）。如果某个用户被确认为入侵者，其对应的细节信息就要生成日志。IP地址可以用正则表达式和`egrep`命令从日志中提取。试图登录的次数就是含有某个用户的日志行数。这个行数可以用`wc`命令获得。IP地址到主机名的映射可以将IP地址作为`host`命令的参数，然后从命令输出中提取。时间范围可以用已经提取到的时间戳来打印。最后，删除脚本使用过的临时文件。

上节的脚本旨在演示一个用于扫描日志并从中生成报告的模型。我们尽力让脚本更短小简单，以避免过于复杂，因此难免存在一些bug。你可以使用更好的处理逻辑来改进该脚本。

8.11 监视远程磁盘的健康情况

网络是由不同用户的多台主机组成。它需要对远程主机的磁盘使用情况实施集中监视。网络的系统管理员得每天记录网络中所有主机的磁盘使用。日志的每一行应该包含日期、主机IP地址、设备、设备容量、占用空间、剩余空间、使用比例、健康状况等细节信息。如果远程主机中的任

意分区使用率超过了80%，那么健康状态应该被设置为ALERT，否则就可以设置为SAFE。这则攻略将演示如何编写一个可以收集网络远程主机详细信息的监视脚本。

8.11.1　预备知识

我们需要从网络中收集每台主机的磁盘使用情况信息，然后写入中央主机的日志文件中。可以将负责收集信息并写入日志的脚本调度为每天的特定时间执行。可以使用SSH来登录远程系统收集磁盘使用情况。

8.11.2　实战演练

首先得在网络中的所有远程主机上设置一个共用账户。这个账户供脚本disklog登录系统使用。我们需要为这个账户配置SSH自动登录（7.8节讲解了自动登录的配置方法）。假设在所有配置了自动登录的远程主机上都有一个叫做test的用户，那么来看看这个shell脚本：

```
#!/bin/bash
#文件名: disklog.sh
#用途: 监视远程系统的磁盘使用情况


logfile="diskusage.log"

if [[ -n $1 ]]
then
  logfile=$1
fi

if [ ! -e $logfile ]
then

  printf "%-8s %-14s %-9s %-8s %-6s %-6s %-6s %s\n" "Date" "IP address" "Device"
"Capacity" "Used" "Free" "Percent" "Status" > $logfile
fi

IP_LIST="127.0.0.1 0.0.0.0"
#提供远程主机IP地址列表

(
for ip in $IP_LIST;
do
  #slynux是用户名，可以按照实际情况进行修改
  ssh slynux@$ip 'df -H' | grep ^/dev/ > /tmp/$$.df

  while read line;
```

```
    do
      cur_date=$(date +%D)
      printf "%-8s %-14s " $cur_date $ip
      echo $line | awk '{ printf("%-9s %-8s %-6s %-6s %-8s",$1,$2,$3,$4,$5); }'

    pusg=$(echo $line | egrep -o "[0-9]+%")
    pusg=${pusg/\%/};
    if [ $pusg -lt 80 ];
    then
      echo SAFE
    else
      echo ALERT
    fi

    done< /tmp/$$.df
  done

) >> $logfile
```

我们可以用cron以固定的间隔来调度脚本执行，例如在crontab中写入以下条目，就可以在每天上午10点运行该脚本：

```
00 10 * * * /home/path/disklog.sh /home/user/diskusg.log
```

执行命令crontab -e，然后添加上面一行。也可以手动执行脚本：

```
$ ./disklog.sh
```

脚本的输出样例如下：

```
slynux@slynux-laptop:~/book$ cat diskusage.log
Date     IP address     Device    Capacity Used   Free   Percent Status
12/15/10 127.0.0.1      /dev/sda1 9.9G     2.4G   7.0G   26%     SAFE
12/15/10 0.0.0.0        /dev/sda1 9.9G     2.4G   7.0G   26%     SAFE
```

8.11.3 工作原理

在脚本disklog.sh中，我们可以提供日志文件路径作为命令行参数，否则脚本使用默认的日志文件。如果日志文件不存在，它会将日志文件头部写入新文件中。-e logfile用来检查文件是否存在。远程主机的IP地址列表被存储在变量IP_LIST中，彼此之间以空格分隔。要确保在IP_LIST中列出的所有远程系统中都有用户test，并且SSH已经配置了自动登录。for循环用来对IP地址进行逐个迭代。通过ssh使用远程命令df -H获取磁盘剩余空间。这项数据被存储在一个临时文件中。while循环用来逐行读取这个文件。利用awk提取并打印数据，同时一并打印的还有日期。用egrep提取使用率，并将%删除以获取使用率的数值部分。检查得到的数值，看是否超过了80。如果不足80，将状态设置为SAFT；如果大于或等于80，则将状态设置为ALERT。打

印出来的所有数据要被重定向到日志文件中。因此代码被放入子shell()中，并将标准输出重定向到日志文件。

8.11.4 参考

9.7节讲解了crontab命令。

8.12 找出系统中用户的活跃时段

考虑一个使用共享主机的Web服务器。每天都会有很多用户登录和注销，用户活动都被记入了服务器的系统日志。这则攻略是一项实践任务：利用系统日志找出每个用户在服务器上停留了多久，并根据其时间长短来进行分级，最后生成一份包含等级、用户名、首次登录时间、末次登录时间、登录次数以及总使用时长等细节信息的报告。让我们看看如何解决这个问题。

8.12.1 预备知识

last命令用来列出系统中有关用户登录会话的细节。这些会话数据被存储在/var/log/wtmp文件中。通过分别累计各用户的会话时间，就能得出他们的总使用时间。

8.12.2 实战演练

研究一下这个用来找出活跃用户并生成报告的脚本：

```
#!/bin/bash
#用户名： active_users.sh
#用途:查找活跃用户

log=/var/log/wtmp

if [[ -n $1 ]];
then
  log=$1
fi

printf "%-4s %-10s %-10s %-6s %-8s\n" "Rank" "User" "Start" "Logins" "Usage hours"

last -f $log | head -n -2   > /tmp/ulog.$$

cat /tmp/ulog.$$ |  cut -d' ' -f1 | sort | uniq> /tmp/users.$$

(
while read user;
do
```

```
  grep ^$user /tmp/ulog.$$ > /tmp/user.$$
  minutes=0

  while read t
  do
    s=$(echo $t | awk -F: '{ print ($1 * 60) + $2 }')
    let minutes=minutes+s
  done< <(cat /tmp/user.$$ | awk '{ print $NF }' | tr -d ')(')

  firstlog=$(tail -n 1 /tmp/user.$$ | awk '{ print $5,$6 }')
  nlogins=$(cat /tmp/user.$$ | wc -l)
  hours=$(echo "$minutes / 60.0" | bc)

  printf "%-10s %-10s %-6s %-8s\n"  $user "$firstlog" $nlogins $hours
done< /tmp/users.$$

) | sort -nrk 4 | awk '{ printf("%-4s %s\n", NR, $0) }'
rm /tmp/users.$$ /tmp/user.$$ /tmp/ulog.$$
```

输出样例如下：

```
$ ./active_users.sh
Rank User       Start      Logins Usage hours
1    easyibaa   Dec 11     531    349
2    demoproj   Dec 10     350    230
3    kjayaram   Dec 9      213    55
4    cinenews   Dec 11     85     139
5    thebenga   Dec 10     54     35
6    gateway2   Dec 11     52     34
7    soft132    Dec 12     49     25
8    sarathla   Nov 1      45     29
9    gtsminis   Dec 11     41     26
10   agentcde   Dec 13     39     32
```

8.12.3 工作原理

在脚本active_users.sh中，需要提供日志文件作为命令行参数，否则就使用默认的日志文件wtmp。命令last -f用来打印日志文件的内容。日志文件的第一列是用户名。我们用cut从中提取第一列，然后用sort和uniq找出不重复的用户。对于每一位用户，用grep找出其对应登录会话的日志行并写入一个临时文件。日志的最后一列是用户登录会话的时长。为了找出用户总的使用时间，需要累加所有的会话时长。使用时间的格式是（小时：秒），需要用一个简单的awk脚本将其转换成分钟。

要提取用户的会话时长，得使用awk命令。要移除括号，得使用tr -d。用<(COMMANDS)操作符将使用时长字符串列表作为标准输入传递给while循环，其作用就相当于文件输入。利用date命令将每一个时长字符串转换成秒数，并累加到变量seconds中。把出现在最后一行的用

8

户的首次登录时间提取出来。登录次数就是日志的行数。要根据总的使用时间来计算每位用户的等级，数据记录以使用时长为键，进行降序排列。用`sort`命令的`-nr`选项指定按照数值逆序排列。`-k4`指定键的列号（即使用时长）。最后，`sort`的输出被传递给`awk`。`awk`命令为每一行添加上行号，这个行号就是每一位用户的等级。

8.13　电源使用的测量与优化

必须时刻关注电源的消耗，尤其对于笔记本电脑、平板电脑等移动设备而言更是如此。Linux系统中有几个工具可以测量电源的使用情况，`powertop`就是其中之一，我们将在本攻略中学习它的用法。

8.13.1　预备知识

`powertop`在多数Linux发布版中都没有被预装，你得使用软件包管理器自行安装。

8.13.2　实战演练

来看看如何使用`powertop`来对电源使用进行测量及优化。

(1) `powertop`的用法相当简单，只需要运行以下命令即可：

```
# powertop
```

随后`powertop`就会开始进行测量，测量结束之后会显示出有关电源使用情况、耗电最多的进程等详细信息，如图8-1所示：

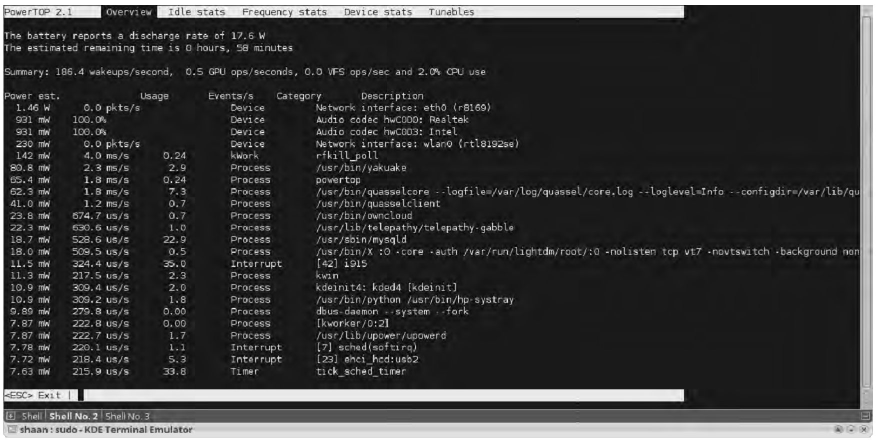


图　8-1

(2) 生成HTML格式的报表：

```
# powertop --html
```

powertop会进行一段时间的测量，然后生成一份默认名称为PowerTOP.html的HTML报表，你可以使用Web浏览器进行浏览。

(3) 优化电源使用：

在powertop运行时，可以使用箭头键切换到Tunables标签。该标签下包含了一系列可由powertop调节的选项，借以降低电源消耗。只需要选中希望调节的选项，按回车键将选项从Bad切换到Good，如图8-2所示。

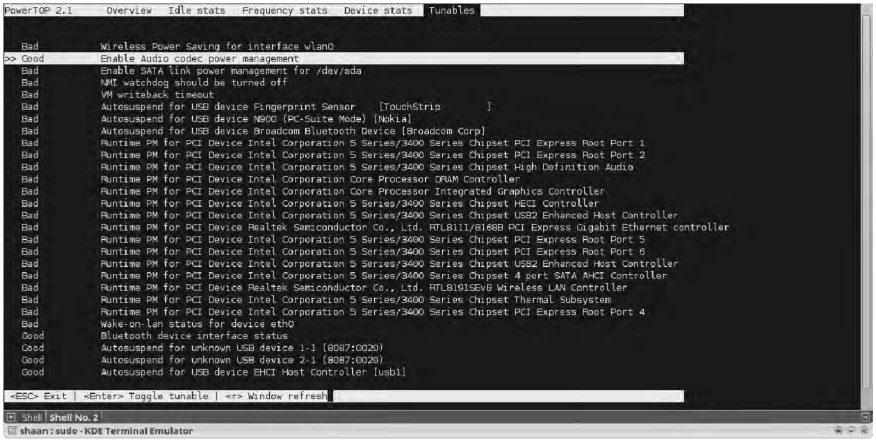


图 8-2

> 如果想要监视可移动设备的电池消耗情况，需要拔掉设备的充电器，使用powertop对电池进行测量。

8.14 监视磁盘活动

监视类工具的名称流行以top（进程监视命令）作为结尾，依照这种命名习惯，监视磁盘I/O的工具就叫做iotop。

8.14.1 预备知识

iotop在多数Linux发布版中都没有被预装，你得使用软件包管理器自行安装。

8.14.2　实战演练

使用iotop进行I/O监视的方法多种多样，在这则攻略中会涉及其中的几种。

(1) 交互式监视：

```
# iotop -o
```

iotop的-o选项只显示出那些正在进行I/O活动的进程。该选项有助于减少输出干扰。

(2) 用于shell脚本的非交互式用法：

```
# iotop -b -n 2
```

这使得iotop打印出两次统计数据，然后退出。如果你希望在shell脚本中获得输出结果并对其进行处理，这种用法就能派上用场了。

(3) 监视特定进程

```
# iotop -p PID
```

使用你希望对其进行监视的进程的PID，iotop就只会输出该进程的统计信息。

> 在大多数现代Linux发布版中，无需先查找PID，然后再提供给iotop。你可以使用pidof命令，将之前的命令写成如下形式：
>
> ```
> # iotop -p 'pidof cp'
> ```

8.15　检查磁盘及文件系统错误

数据是计算机系统中最重要的东西。因而对存储在物理介质上数据的一致性进行监视自然也就很重要了。

8.15.1　预备知识

我们将会使用标准工具fsck来检查文件系统错误。该命令应该已经预装在所有的Linux发布版中了。如果没有预装，请使用软件包管理器自行安装。

8.15.2　实战演练

来看看如何使用fsck的各种选项对文件系统错误进行检查和修复。

(1) 要检查分区或文件系统的错误，只需要将路径作为fsck的参数：

```
# fsck /dev/sdb3
fsck from util-linux 2.20.1
```

```
e2fsck 1.42.5 (29-Jul-2012)
HDD2 has been mounted 26 times without being checked, check forced.
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
HDD2: 75540/16138240 files (0.7% non-contiguous), 48756390/64529088 blocks
```

(2) 检查/etc/fstab中所配置的所有文件系统：

```
# fsck -A
```

该命令会依次检查/etc/fstab中列出的文件系统。`fstab`文件对磁盘及其挂载点之间的映射关系进行了配置，以便于更便捷地挂载文件系统。这也使得在系统启动的过程中能够挂载某些文件系统。

(3) 指定`fsck`自动修复错误，无需询问是否进行修复：

```
# fsck -a /dev/sda2
```

(4) 模拟`fsck`要执行的操作：

```
# fsck -AN
fsck from util-linux 2.20.1
[/sbin/fsck.ext4 (1) -- /] fsck.ext4 /dev/sda8
[/sbin/fsck.ext4 (1) -- /home] fsck.ext4 /dev/sda7
[/sbin/fsck.ext3 (1) -- /media/Data] fsck.ext3 /dev/sda6
```

该命令会打印出`fsck`将要执行什么样的操作（检查所有的文件系统）。

8.15.3 工作原理

`fsck`不过是一个前端程序而已，对于各类文件系统而言，都有其针对性的`fsck`程序。当执行`fsck`时，它会自动检测文件系统类型，然后运行对应的`fsck.fstype`命令，其中`fstype`是文件系统的类型。如果我们在`ext4`文件系统上执行`fsck`，它最终会调用`fsck.ext4`命令。

正因为如此，你会发现`fsck`本身只支持所有这些特定文件系统工具所共有的选项。要查找更详细的选项，请查找特定工具（如`fsck.ext4`）的手册页。

如果怀疑文件系统受损，需要使用`fsck`进行修复时，对`fsck`操作进行模拟则不无益处。因为必须确保`fsck`不会执行我们不希望进行的操作。比如说，`fsck`可能会将某些扇区标为坏扇区，但我们却希望从中恢复数据。在这种情况下，我们只让`fsck`做一个演习，打印出所要进行的操作，而不是真正进行操作。

第 9 章

管理重任

9

本章内容

- 收集进程信息
- 杀死进程以及发送或响应信号
- 向用户终端发送消息
- 采集系统信息
- 使用/proc采集信息
- 使用cron进行调度
- 从Bash中读写MySQL数据库
- 用户管理脚本
- 图像文件的缩放及格式转换
- 在终端下进行截屏
- 管理多个终端

9.1 简介

GNU/Linux的生态系统是由运行的程序、服务、所连接的设备、文件系统、用户等组成的。系统管理的主要目的在于对整个系统形成一个概观并对操作系统进行整体上的管理。我们应该掌握用于系统信息收集及资源管理的常用命令以及实践用法，这也有助于编写脚本和使用自动化工具。本章将介绍一些此类工具的用法。

9.2 收集进程信息

进程是程序的运行实例（running instance）。运行在一台计算机中的多个进程都被分配了一个称为**进程ID**（PID）的唯一标识数字。同一个程序的多个实例可以同时运行，但是它们的PID却互不相同。进程包括多种属性，例如拥有该进程的用户、进程使用的内存数量、进程占用的CPU时间等。这则攻略将学习如何收集有关进程的信息。

9.2.1 预备知识

和进程管理相关的重要命令是top、ps和pgrep。让我们看看如何收集进程信息。

9.2.2 实战演练

ps是收集进程信息的重要工具。它提供的信息包括：拥有进程的用户、进程的起始时间、进程对应的命令行路径、PID、进程所属的终端（TTY）、进程使用的内存、进程占用的CPU等。例如：

```
$ ps
  PID TTY          TIME CMD
 1220 pts/0    00:00:00 bash
 1242 pts/0    00:00:00 ps
```

ps命令通常结合一系列参数使用。如果不使用任何参数，ps将显示运行在当前终端（TTY）中的进程。第一列显示进程ID（PID），第二列是TTY（终端），第三列是进程启动后过去的时间，最后一列是CMD（进程所对应的命令）。

为了包含更多的信息，可以使用-f（表示full）来显示多列，如下所示：

```
$ ps -f
UID        PID  PPID  C STIME TTY          TIME CMD
slynux    1220  1219  0 18:18 pts/0    00:00:00 -bash
slynux    1587  1220  0 18:59 pts/0    00:00:00 ps -f
```

上面的ps命令没有什么用处，因为它没有提供除当前终端外的任何进程信息。要获取运行在系统中的每个进程的信息，使用选项 -e（every）。选项-ax（all）也可以生成同样的输出。

> 选项-x和-a用来解除ps默认设置的TTY限制。通常使用不带参数的ps命令，以便只打印出其所属终端上的进程。

运行如下命令之一：ps -e，ps -ef，ps -ax或ps -axf。

```
$ ps -e | head
  PID TTY          TIME CMD
1 ?        00:00:00 init
2 ?        00:00:00 kthreadd
3 ?        00:00:00 migration/0
4 ?        00:00:00 ksoftirqd/0
5 ?        00:00:00 watchdog/0
6 ?        00:00:00 events/0
7 ?        00:00:00 cpuset
8 ?        00:00:00 khelper
9 ?        00:00:00 netns
```

输出列表很长。我们使用head进行了过滤，所以只列出了前10项。

ps命令支持显示包括进程名及PID在内的多种信息。ps默认在不同的列中显示这些信息，

其中的大多数都没什么用处。可以用-o来指定想要显示的列，以便只打印出我们需要的内容。与进程相关的参数可以通过与此参数对应的命令选项指定。参数列表以及-o的用法在接下来会进行讨论。

用ps显示所需要的输出列：

```
$ ps [OTHER OPTIONS] -o parameter1,parameter2,parameter3 ..
```

> -o的参数以逗号操作符（，）作为定界符。值得注意的是，逗号操作符与它分隔的参数之间是没有空格的。在大多数情况下，选项-o都是和选项-e结合使用的（-oe），因为它需要列出运行在系统中的每一个进程。但是如果-o需要使用某些过滤器，例如列出特定用户拥有的进程，那么就不再使用-e。-e和过滤器结合使用没有任何实际效果，依旧会显示所有的进程。

示例如下，其中comm表示COMMAND，pcpu表示CPU占用率：

```
$ ps -eo comm,pcpu | head
COMMAND         %CPU
init             0.0
kthreadd         0.0
migration/0      0.0
ksoftirqd/0      0.0
watchdog/0       0.0
events/0         0.0
cpuset           0.0
khelper          0.0
netns            0.0
```

9.2.3　工作原理

选项-o可以使用不同的参数，这些参数及其描述如表9-1所示。

表　9-1

参　　数	描　　述
pcpu	CPU占用率
pid	进程ID
ppid	父进程ID
pmem	内存使用率
comm	可执行文件名

（续）

参　数	描　述
cmd	简单命令[①]
user	启动进程的用户
nice	优先级
time	累计的CPU时间
etime	进程启动后流逝的时间
tty	所关联的TTY设备
euid	有效用户ID
stat	进程状态

9.2.4　补充内容

让我们看看其他一些进程控制命令的用法。

1. top

top对于系统管理员来说是一个极为重要的命令。它默认会输出一个占用CPU最多的进程列表。输出结果每隔几秒就会更新。该命令的用法如下所述。

```
$ top
```

除了若干个占用CPU最多的进程外，还会显示进程相关的一些其他参数，如图9-1所示。

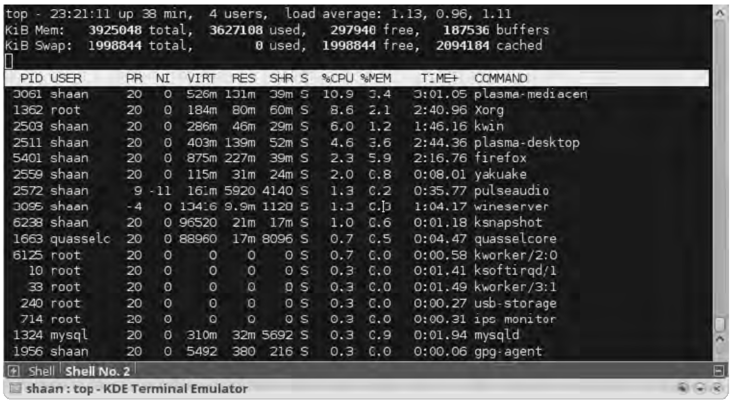


图　9-1

---

① 简单命令是我们平时使用最频繁的一类命令。它是由空白字符分隔的一系列单词，以shell控制操作符作为结尾。第一个单词指定要执行的命令，余下的单词作为命令参数。shell控制操作符可以是换行符，或者是`||`,`&&`,`&`,`;`,`;;`,`|`,`|&`,`(`,`)`。

2. 根据参数对ps输出进行排序

可以用--sort将ps命令的输出根据特定的列进行排序。在参数前加上+（升序）或-（降序）来指定排序方式：

```
$ ps [OPTIONS] --sort -paramter1,+parameter2,parameter3..
```

例如，要列出占用CPU最多的10个进程：

```
$ ps -eo comm,pcpu --sort -pcpu | head
COMMAND          %CPU
Xorg              0.1
hald-addon-stor   0.0
ata/0             0.0
scsi_eh_0         0.0
gnome-settings-   0.0
init              0.0
hald              0.0
pulseaudio        0.0
gdm-simple-gree   0.0
```

进程依据CPU占用率进行降序排序，用head命令提取前10个进程。

我们可以用grep从ps的输出中提取与给定进程名或其他参数相关的条目。要找出与Bash进程相关的条目，可以使用：

```
$ ps -eo comm,pid,pcpu,pmem | grep bash
bash              1255  0.0  0.3
bash              1680  5.5  0.3
```

3. 找出给定命令名所对应的进程ID

假设某个命令有多个实例正在运行，我们可能需要识别这些进程的PID。该信息可以使用ps或pgrep命令得到。按照下面的方式使用ps：

```
$ ps -C COMMAND_NAME
```

或者

```
$ ps -C COMMAND_NAME -o pid=
```

用户自定义格式指示符-o先前已经讲解过了。但是这里你可以看到pid后面加上了 =，这会去掉ps 输出中 PID一列的列名。在参数后加上=就可以移除列名。例如：

```
$ ps -C bash -o pid=
 1255
 1680
```

这条命令列出了所有Bash进程的进程ID。

除此之外，还有一个很方便的工具pgrep。你可以用它获得特定命令的进程ID列表。例如：

```
$ pgrep COMMAND
$ pgrep bash
1255
1680
```

> pgrep只需要命令名的一部分作为输入参数来提取Bash命令，诸如pgrep ash或pgrep bas都能够奏效。但是ps需要你输入命令准确的全名。

pgrep可以接受多个输出过滤选项。如果不使用换行符作为定界符，而是要自行指定可以像下面这样：

```
$ pgrep COMMAND -d DELIMITER_STRING
$ pgrep bash -d ":"
1255:1680
```

指定进程的用户（拥有者）列表：

```
$ pgrep -u root,slynux COMMAND
```

其中root和slynux都是用户名。返回所匹配的进程数量：

```
$ pgrep -c COMMAND
```

4. 根据真实用户或ID以及有效用户或ID过滤ps输出

可以用ps根据指定的真实/有效用户名或ID对进程进行分组。指定的参数可以用来过滤ps的输出：通过检查每一个输出条目是否属于参数列表中指定的有效用户或真实用户，并只显示匹配的条目。实现方法如下：

- 用 -u EUSER1,EUSER2 …，指定有效用户列表；
- 用 -U RUSER1,RUSER2 …，指定真实用户列表。

例如：

```
$ ps -u root -U root -o user,pcpu
```

该命令会显示以root作为有效用户ID和真实用户ID的所有进程，以及用户、CPU占用率列。

> 在大多数情况下，-o都是和-e结合成-eo的形式。但是当使用过滤器的时候，-o应该像上面那样单独使用。

5. 用TTY过滤**ps**输出

可以通过指定进程所属的TTY选择ps的输出。用选项 -t指定TTY列表：

```
$ ps -t TTY1, TTY2 ..
```

例如：

```
$ ps -t pts/0,pts/1
  PID TTY          TIME CMD
 1238 pts/0    00:00:00 bash
 1835 pts/1    00:00:00 bash
 1864 pts/0    00:00:00 ps
```

6. 进程线程的相关信息

通常与进程线程相关的信息在ps输出中是看不到的。我们可以用选项 -L 在ps输出中显示线程的相关信息。这会显示出两列：NLWP和NLP。NLWP是进程的线程数量，NLP是ps输出中每个条目的线程ID。例如：

```
$ ps -eLf
```

或者

```
$ ps -eLf --sort -nlwp | head
UID        PID  PPID   LWP  C NLWP STIME TTY          TIME CMD
root       647     1   647  0   64 14:39 ?        00:00:00 /usr/sbin/
console-kit-daemon --no-daemon
root       647     1   654  0   64 14:39 ?        00:00:00 /usr/sbin/
console-kit-daemon --no-daemon
root       647     1   656  0   64 14:39 ?        00:00:00 /usr/sbin/
console-kit-daemon --no-daemon
root       647     1   657  0   64 14:39 ?        00:00:00 /usr/sbin/
console-kit-daemon --no-daemon
root       647     1   658  0   64 14:39 ?        00:00:00 /usr/sbin/
console-kit-daemon --no-daemon
root       647     1   659  0   64 14:39 ?        00:00:00 /usr/sbin/
console-kit-daemon --no-daemon
root       647     1   660  0   64 14:39 ?        00:00:00 /usr/sbin/
console-kit-daemon --no-daemon
root       647     1   662  0   64 14:39 ?        00:00:00 /usr/sbin/
console-kit-daemon --no-daemon
root       647     1   663  0   64 14:39 ?        00:00:00 /usr/sbin/
console-kit-daemon --no-daemon
```

该命令列出了线程数最多的10个进程。

7. 指定输出宽度以及所要显示的列

我们可以使用用户自定义的输出格式指示符-o来指定在ps输出中所要显示的列。另一种指

定输出格式的方法是使用标准选项。可以按照你自己的使用方式来进行应用。尝试以下选项:

- ❑ -f ps -ef
- ❑ u ps -e u
- ❑ ps ps -e w（w表示宽松输出）

8. 显示进程的环境变量

了解某个进程依赖哪些环境变量，这类信息我们通常都用得着。进程的运行方式可能极其依赖某组环境变量。我们可以利用环境变量调试并修复与进程相关的问题。

要在ps的输出条目中同时列出环境变量，可以使用：

```
$ ps -eo cmd e
```

例如：

```
$ ps -eo pid,cmd  e | tail -n 3
 1162 hald-addon-acpi: listening on acpid socket /var/run/acpid.socket
 1172 sshd: slynux [priv]
 1237 sshd: slynux@pts/0
 1238 -bash USER=slynux LOGNAME=slynux HOME=/home/slynux
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
MAIL=/var/mail/slynux SHELL=/bin/bash SSH_CLIENT=10.211.55.2 49277 22
SSH_CONNECTION=10.211.55.2 49277 10.211.55.4 22 SSH_TTY=/dev/pts/0 TERM=xterm-color
LANG=en_IN XDG_SESSION_COOKIE=d1e96f5cc8a7a3bc3a0a73e44c95121a-1286499339.
592429-1573657095
```

这种跟踪环境方法的用途之一就是解决apt-get软件包管理器的故障。如果你是用HTTP代理连接Internet，可能得设置环境变量http_proxy=host:port。如果你在脚本中忘记设置该环境变量，那么apt-get就没法选择代理服务器并将返回一个错误。这时就得查看环境变量，查找问题所在。

我们可能需要借助crontab这类调度工具来使一些应用程序能够自动运行。但是这类应用也许要依赖某些环境变量。假设需要在某个特定时间打开一个窗口化的GUI应用程序。我们用crontab将其安排到指定的时间，但结果却事与愿违：

```
00 10 * * * /usr/bin/windowapp
```

这是因为窗口化的应用程序总是依赖于环境变量DISPLAY。要想获得所需的环境变量，首先手动运行windowapp，然后运行ps -C windowapp -eo cmd e。找出对应的环境变量，将其添加到crontab中的命令名之前，这个问题就可以搞定了：

```
00 10 * * * DISPLAY=:0 /usr/bin/windowapp
```

这里的DISPLAY=:0 是从ps输出中获取的。

9. which、whereis、file、whatis与平均负载

有一些命令可以用来探究其他的命令。让我们来看一下。

- **which**

  which命令用来找出某个命令的位置。我们在终端输入命令时无需知道对应的可执行文件位于何处。

  当输入命令时，终端会在一组位置中查找这个命令，如果能够找到，那么就执行该可执行文件。这一组位置由环境变量PATH指定。例如：

```
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
```

  我们可以导出PATH并添上我们自己的命令搜索位置。如果要将/home/slynux/bin添加到PATH中，可以使用以下命令：

```
$ export PATH=$PATH:/home/slynux/bin
# /home/slynux/bin被添加到PATH中
```

  which命令会输出作为其参数的命令所在的位置。例如：

```
$ which ls
/bin/ls
```

- **whereis**

  whereis与which命令类似，但它不仅返回命令的路径，还能够打印出其对应的命令手册的位置以及命令源代码的路径（如果有的话）。例如：

```
$ whereis ls
ls: /bin/ls /usr/share/man/man1/ls.1.gz
```

- **file**

  file命令是一个既有趣又使用频繁的命令，可用来确定文件的类型：

```
$ file FILENAME
```

  该命令会打印出与该文件类型相关的细节信息。

  例如：

```
$ file /bin/ls
/bin/ls: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked
(uses shared libs), for GNU/Linux 2.6.15, stripped
```

- **whatis**

  whatis命令会输出作为参数的命令的简短描述信息。这些信息是从命令手册中解析得来的。例如：

```
$ whatis ls
ls (1)              - list directory contents
```

> **apropos**
>
> 有时候我们需要搜索和某个单词相关的命令是否存在。那么可以搜索包含该字符串命令的手册页。为此，我们可以使用以下命令：
>
> `apropos COMMAND`

- ❑ 平均负载

  平均负载是系统总负载量的一个重要参数。它指明了系统中可运行进程总量的平均值。平均负载由三个值来指定，第一个值指明了1分钟内的平均值，第二个值指明了5分钟内的平均值，第三个值指明了15分钟内的平均值。

  这三个值可以通过运行`uptime`获得。例如：

```
$ uptime
 12:40:53 up  6:16,  2 users,  load average: 0.00, 0.00, 0.00
```

9.2.5 参考

9.7节讲解了如何调度任务。

9.3 杀死进程以及发送或响应信号

终结进程是我们通常都会碰到的事儿，包括需要终结某个程序的所有实例。命令行提供了多种用于终结程序的方法。在类Unix环境中与进程相关的一个重要概念就是信号。信号是一种进程间通信机制，它用来中断运行中的进程以执行某些操作。终止程序也是通过使用信号技术来实现的。这则攻略介绍了信号及其用法。

9.3.1 预备知识

信号是Linux中的一种进程间通信机制。我们可以使用特定的信号来中断进程。每一种信号都同一个整数值相关联。当进程接收到一个信号时，它会通过执行对应的信号处理程序（signal handler）来进行响应。在shell脚本中同样可以发送、接收并响应信号。`kill`是用于终止进程的信号。像Ctrl+C、Ctrl+Z这种事件也会发送各自的信号。`kill`命令可用来向进程发送信号，而`trap`命令用来处理所接收的信号。

9.3.2 实战演练

(1) 列出所有可用的信号：

```
$ kill -l
```

该命令会打印出信号编号（signal number）以及对应的信号名称。

(2) 终止进程：

```
$ kill PROCESS_ID_LIST
```

`kill`命令默认发出一个`TERM`信号。进程ID列表使用空格作为进程ID之间的定界符。

(3) 要通过`kill`命令向进程发送指定的信号，可以使用：

```
$ kill -s SIGNAL PID
```

参数`SIGNAL`要么是信号名称，要么是信号编号。尽管有很多信号可用于不同的目的，但经常用到的其实只有那么几个，具体如下所示。

- ❑ `SIGHUP 1`——对控制进程或终端的终结进行挂起检测（hangup detection）。
- ❑ `SIGINT 2`——当按下Ctrl + C时发送该信号。
- ❑ `SIGKILL 9`——用于强行杀死进程。
- ❑ `SIGTERM 15`——默认用于终止进程。
- ❑ `SIGTSTP 20`——当按下Ctrl + Z时发送该信号。

(4) 我们经常要强行杀死进程，可以使用：

```
$ kill -s SIGKILL PROCESS_ID
```

或者

```
$ kill -9 PROCESS_ID
```

9.3.3 补充内容

让我们看看其他一些用于终止以及向进程发送信号的命令。

1. 相关的kill命令

`kill`命令以进程ID作为参数。在`kill`命令系列中还有其他命令可以接受命令名作为参数，向对应的进程发送信号。

`killall`命令通过命令名终止进程：

```
$ killall process_name
```

通过名称向进程发送信号：

```
$ killall -s SIGNAL process_name
```

通过名称强行杀死进程：

```
$ killall -9 process_name
```

例如：

```
$ killall -9 gedit
```

通过名称以及所属用户名指定进程：

```
$ killall -u USERNAME process_name
```

如果需要在杀死进程前进行确认，可以使用killall的-i选项。

pkill命令和kill命令类似，不过默认情况下pkill接受的是进程名，而非进程ID。例如：

```
$ pkill process_name
$ pkill -s SIGNAL process_name
```

SIGNAL是信号编号。pkill不支持信号名称。pkill提供了很多和kill相同的选项。要了解更多详细信息，请参阅pkill的命令手册。

2. 捕捉并响应信号

trap命令在脚本中用来为信号分配信号处理程序。一旦使用trap将某个函数分配给一个信号，那么当脚本运行收到该信号时，其对应的函数就会开始执行。

命令语法如下：

```
trap 'signal_handler_function_name' SIGNAL LIST
```

SIGNAL LIST以空格分隔，它可以是信号编号或者信号名称。

下面是一个能够响应信号SIGINT的shell脚本：

```
#/bin/bash
#文件名：sighandle.sh
#用途：信号处理程序

function handler()
{
  echo Hey, received signal : SIGINT
}

echo My process ID is $$
#$$是一个特殊变量，它可以返回当前进程/脚本的进程ID
trap 'handler' SIGINT
```

```
#handler是信号SIGINT的信号处理程序的名称

while true;
do
  sleep 1
done
```

在终端中运行这个脚本。当脚本运行时，如果按Ctrl+C，就会显示一条消息，这是通过执行与信号关联的信号处理程序实现的。Ctrl+C会发出一个`SIGINT`信号。

通过使用一个无限循环`while`来保持进程运行。这样就可以使它能够响应另一个进程以异步方式发送的信号。用来保持进程一直处于活动状态的循环通常称为**事件循环**（event loop）。

我们可以用`kill`命令以及脚本的进程ID向脚本发送信号：

```
$ kill -s SIGINT PROCESS_ID
```

上面脚本的`PROCESS_ID`在执行时会被打印出来。或者也可以用`ps`命令找出脚本的进程ID。

如果没有为信号指定信号处理程序，那么将会调用由操作系统默认分配的信号处理程序。一般来说，按下Ctrl+C会终止程序，因为这是操作系统提供的处理程序的默认行为。不过这里我们自定义的信号处理程序指定了在接收到信号后所执行的操作。

通过`trap`命令，我们能够为任何可用的信号（`kill -l`）定义处理程序，也可以为多个信号指定单个信号处理程序。

9.4　向用户终端发送消息

系统管理员可能需要向网络中所有用户或特定用户的终端发送消息。这则攻略将指导你如何完成这项任务。

9.4.1　预备知识

`wall`命令用来向当前所有登录用户的终端写入消息。它可以将消息传递给一台服务器中所有的登录用户或是多台分散主机中的用户。给所有的用户发送消息未必总是靠谱。在Linux系统中，终端是作为设备存在的。因此那些打开的终端在/dev/pts/中都会有对应的设备节点文件。向特定的设备写入数据将会在对应的终端中显示出消息。

9.4.2　实战演练

向终端中所有的当前登录用户发送广播消息：

```
$ cat message | wall
```

或者

```
$ wall< message
Broadcast Message from slynux@slynux-laptop
        (/dev/pts/1) at 12:54 ...

This is a message
```

消息概要（message outline）会显示谁（哪个用户、哪台主机）发送了这则消息。如果其他用户发送了消息，只有在“写入消息”选项启用的情况下该消息才会显示在当前终端中。在绝大多数发布版中，“写入消息”选项都是默认启用的。如果消息的发送者是root用户，那么不管“写入消息”选项是否启用，消息都会显示出来。

要允许写入消息，可以使用：

```
$ mesg y
```

要禁止写入消息，可以使用：

```
$ mesg n
```

让我们写一个给指定用户终端发送消息的脚本：

```
#/bin/bash
#文件名：message_user.sh
#用途： 用于向指定用户的终端发送消息的脚本
USERNAME=$1

devices=`ls /dev/pts/* -l | awk '{ print $3,$10 }' | grep $USERNAME | awk '{ print $2 }'`
for dev in $devices;
do
  cat /dev/stdin > $dev
done
```

运行脚本：

```
./message_user.sh USERNAME < message.txt
#通过stdin传递消息，USERNAME作为参数
```

输出如下：

```
$ cat message.txt
A message to slynux. Happy Hacking!
#./message_user.sh slynux  < message.txt
#因为消息要发送给指定的用户，因此要以root用户身份运行message_user.sh
```

这时`slynux`的终端就会接收到该消息。

9.4.3　工作原理

目录/dev/pts中包含着与每一位系统终端中登录用户所对应的字符设备。我们可以通过查看`ls -l`输出设备文件的属主来得知谁登入了哪个终端。这些信息可以用`awk`来获得，然后借助`grep`提取出对应于指定用户的行。用户名作为脚本的首个参数被存储在变量`USERNAME`中。`$devices`中包含了给定用户的终端列表，该列表通过`for`循环来进行迭代。/dev/stdin包含传递给当前进程的标准输入数据，这些数据被重定向到对应的终端设备（TTY）。

9.5　采集系统信息

从命令行中收集当前系统信息对于记录系统数据来说非常重要。各种系统信息包括主机名、内核版本、Linux发布版名称、CPU信息、内存信息、磁盘分区信息等。这则攻略将为你演示在Linux中采集系统信息的各种途径。

实战演练

(1) 打印当前系统的主机名：

```
$ hostname
```

或者

```
$ uname -n
```

(2) 打印Linux内核版本、硬件架构等详细信息：

```
$ uname -a
```

(3) 打印内核发行版本：

```
$ uname -r
```

(4) 打印主机类型：

```
$ uname -m
```

(5) 打印CPU相关信息：

```
$ cat /proc/cpuinfo
```

获取处理器名称：

```
$ cat /proc/cpuinfo | sed -n 5p
```

cpuinfo的第5行包含处理器的名称。

(6) 打印内存的详细信息：

```
$ cat /proc/meminfo
```

打印系统可用内存总量：

```
$ cat /proc/meminfo  | head -1
MemTotal:        1026096 kB
```

(7) 列出系统的分区信息：

```
$ cat /proc/partitions
```

或者

```
$ fdisk -l  #如果没有输出，切换到root用户执行该命令
```

(8) 获取系统的详细信息：

```
$ lshw      #建议以root用户来执行
```

9.6 使用 proc 采集信息

在GNU/Linux操作系统中，/proc是一个在内存中的伪文件系统（pseudo filesystem）。它的引入是为了提供一个可以从用户空间读取系统参数的接口。我们能够从中获取到大量的信息。下面来看看如何使用它。

实战演练

如果查看/proc，你会发现有很多文件和目录。其中的一些我们在本章的其他攻略中已经讲解过了。你可以对/proc及其子目录下的文件使用cat来获取信息。所有内容都是易读的格式化文本。

系统中每一个运行的进程在/proc中都有一个对应的目录。目录名和进程ID相同。

以Bash为例，它的进程ID是4295（pgrep bash），那么就会有一个对应的目录/proc/4295。进程对应的目录中包含了大量有关进程的信息。/proc/PID中一些重要的文件如下所示。

- environ：包含与进程相关的环境变量。使用cat /proc/4295/environ，可以显示所有传递给该进程的环境变量。
- cwd：是一个到进程工作目录（working directory）的符号链接。
- exe：是一个到当前进程所对应的可执行文件的符号链接。

```
$ readlink /proc/4295/exe
/bin/bash
```

- ❑ fd：包含了进程所使用的文件描述符。

9.7　用 cron 进行调度

我们经常会需要安排脚本在某个时间或每隔一段时间来运行。GNU/Linux系统包含了各种可用于任务调度的工具。cron就是其中之一，它通过守护进程cron使得任务能够按照固定的时间间隔在系统后台自动运行。cron利用的是一个叫做cron表的文件，这个文件中存储了需要执行的脚本或命令的调度列表以及执行时间。一个常见的用法是设置在免费时段（一些ISP提供免费使用时间，通常是在午夜），从Internet上进行下载。用户完全不需要在夜里熬红双眼等待下载。只需要编写一个cron条目，然后调度下载即可。你也可以安排当免费时段结束后自动断开Internet连接并关机。

9.7.1　预备知识

所有的GNU/Linux发布版默认都包含了cron调度工具。只要在cron表中写入条目，对应的命令就会在指定的时间执行。crontab命令用来添加作业（job）。cron表不过是一个简单的文本文件而已，每位用户都有自己的副本。

9.7.2　实战演练

要想进行任务调度，我们得知道cron表的格式。一项cron作业指定了需要执行的脚本或命令的路径以及执行时间。

(1) 创建一项cron作业，在每天中每小时的第2分钟执行脚本test.sh：

```
02 * * * * /home/slynux/test.sh
```

(2) 在每天的第5、6、7小时执行脚本：

```
00 5,6,7 * * /home/slynux/test.sh
```

(3) 在周日的每个小时执行脚本script.sh：

```
00 */12 * * 0 /home/slynux/script.sh
```

(4) 在每天凌晨2点关闭计算机：

```
00 02 * * * /sbin/shutdown -h
```

(5) 现在，让我们看看如何调度一项cron作业。利用crontab命令进行调度的方法有很多种。

使用选项 -e来编辑cron表：

```
$ crontab -e
02 02 * * * /home/slynux/script.sh
```

输入crontab -e后，会打开默认的文本编辑器（通常是vi）供用户输入cron作业并保存。该cron作业将会在指定的时间被调度执行。

(6) 如果我们在脚本中调用crontab进行任务调度，那么有另外两种方法可供使用。

① 创建一个文本文件（例如task.cron），并写入cron作业。将文件名作为命令参数，运行crontab：

```
$ crontab task.cron
```

② 在行内（inline）指定cron作业，而无需创建单独的文件。例如：

```
crontab<<EOF
02 * * * * /home/slynux/script.sh
EOF
```

cron作业需要写在crontab<<EOF和EOF之间。

9.7.3 工作原理

cron表中的每一个条目都由6部分组成，并按照下列顺序排列：

- 分钟（0～59）
- 小时（0～23）
- 天（1～31）
- 月份（1～12）
- 工作日（0～6）
- 命令（在指定时间执行的脚本或命令）

前5部分指定了开始执行某个命令实例的时间。还有其他一些选项也可用来指定调度时间。

星号（*）指定命令应该在每个时间段执行。也就是说，如果*是写在cron作业中的小时字段中，那么命令就会每小时执行一次。与此类似，如果你希望在某个特定时段执行命令，那么就在对应的时间字段中指定时段，并用逗号分隔（例如要在5分钟和10分钟时运行命令，那就在分钟字段中输入"5,10"）。还有另一个不错的选项可以让我们以特定的时间间隔运行命令。在分钟字段使用*/5，可以每5分钟运行一次命令。这个技巧可以用在任何时间字段。一个cron表条目

是由一项或多项cron作业组成的。cron表条目中的每一行都是一项作业。

执行cron作业所使用的权限同执行crontab命令所使用的权限相同。如果你需要执行要求更高权限的命令，例如关闭计算机，那么就要以root用户身份执行crontab。

在cron作业中指定的命令需要使用完整路径。这是因为执行cron作业时的环境与终端所使用的环境不同，环境变量PATH可能都没有设置。如果命令运行时需要设置某些环境变量，就应该明确地进行设定。

9.7.4　补充内容

crontab命令还包括其他选项。让我们看看其中一部分。

1. 指定环境变量

很多命令需要正确的设置环境变量才能够运行。我们可以在用户的cron表中插入一行变量赋值语句来设置环境变量。

例如，如果你使用的是代理服务器连接互联网，要调度某个需要使用互联网的命令，就得设置HTTP代理环境变量http_proxy，可以用下面的方法来完成：

```
crontab<<EOF
http_proxy=http://192.168.0.3:3128
00 * * * * /home/slynux/download.sh
EOF
```

2. 在系统启动时运行命令

有时候需要在系统启动时运行特定的命令。实现这一需求的方法有很多，cron便是其中之一。（其他一些方法是将命令添加到/etc/rc.d中，但这并不能保证在所有的发布版中都行得通。）

要在启动时运行命令，请将下面一行加入crontab：

```
@reboot command
```

这样就会以你所具有的用户身份来运行指定的命令。如果需要以root用户的身份运行命令，需要编辑root用户的crontab。

3. 查看cron表

我们可以用选项-l列出cron表中现有的内容：

```
$ crontab -l
02 05 * * * /home/user/disklog.sh
```

crontab -l会列出当前用户cron表中的已有条目。

我们也可以通过选项 -u来查看其他用户的cron表：

```
$ crontab -l -u slynux
09 10 * * * /home/slynux/test.sh
```

当使用选项-u时，你必须作为root用户以获取更高的权限。

4. 删除cron表

可以使用选项-r删除当前用户的cron表：

```
$ crontab -r
```

要删除其他用户的cron表，可以使用：

```
# crontab -u slynux -r
```

这需要以root用户身份以获得更高的权限。

9.8 从 Bash 中读写 MySQL 数据库

MySQL是一款应用广泛的数据库管理系统。以PHP、Python、C++等语言编写的应用程序通常使用它作为存储系统。从shell脚本中访问并处理MySQL数据库也很有意思。我们可以编写脚本将文本文件或CSV（Comma Separated Value，逗号分隔值文件）的内容写入数据表，与MySQL进行交互来读取、处理数据。例如，可以从shell脚本中执行查询语句来读取存储在留言板程序的数据库中的所有电子邮件地址。在这则攻略中，我们会看到如何从Bash中读写MySQL数据库。假设现在有以下问题。

我有一个包含学生详细信息的CSV文件。我需要将文件的内容插入到一个数据表中。要保证为每一个系生成一个单独的排名列表。

9.8.1 预备知识

要处理MySQL数据库，系统中必须安装mysql-server和mysql-client软件包。Linux发布版中默认并没有包含这些工具。由于MySQL要使用用户名和密码进行认证，在安装MySQL服务器时还得设置用户名和密码。

9.8.2 实战演练

前面提出的问题可以用sort、awk等工具解决，或者用一个SQL数据库的数据表也可以搞定。

我们接下来要编写3个脚本，分别用于创建数据库及数据表、向数据表中插入学生数据、从数据表中读取并显示处理过的数据。

创建数据库及数据表的脚本如下：

```
#!/bin/bash
#文件名：create_db.sh
#用途：创建MySQL数据库和数据表

USER="user"
PASS="user"

mysql -u $USER -p$PASS <<EOF 2> /dev/null
CREATE DATABASE students;
EOF

[ $? -eq 0 ] && echo Created DB || echo DB already exist
mysql -u $USER -p$PASS students <<EOF 2> /dev/null
CREATE TABLE students(
id int,
name varchar(100),
mark int,
dept varchar(4)
);
EOF

[ $? -eq 0 ] && echo Created table students || echo Table students already exist

mysql -u $USER -p$PASS students <<EOF
DELETE FROM students;
EOF
```

将数据插入数据表的脚本如下：

```
#!/bin/bash
#文件名：write_to_db.sh
#用途：从CSV中读取数据并写入MySQLdb

USER="user"
PASS="user"

if [ $# -ne 1 ];
then
  echo $0 DATAFILE
  echo
  exit 2
fi

data=$1
```

```
while read line;
do

  oldIFS=$IFS
  IFS=,
  values=($line)
  values[1]="\"`echo ${values[1]} | tr ' ' '#' `\""
  values[3]="\"`echo ${values[3]}`\""


  query=`echo ${values[@]} | tr ' #' ', ' `
  IFS=$oldIFS

  mysql -u $USER -p$PASS students <<EOF
INSERT INTO students VALUES($query);
EOF

done< $data
echo Wrote data into DB
```

查询数据库的脚本如下：

```
#!/bin/bash
#文件名：read_db.sh
#用途:从数据库中读取数据

USER="user"
PASS="user"

depts=`mysql -u $USER -p$PASS students <<EOF | tail -n +2
SELECT DISTINCT dept FROM students;
EOF`

for d in $depts;
do

echo Department : $d

result="`mysql -u $USER -p$PASS students <<EOF
SET @i:=0;
SELECT @i:=@i+1 as rank,name,mark FROM students WHERE dept="$d" ORDER BY mark DESC;
EOF`"

echo "$result"
echo

done
```

作为输入的CSV文件（studentdata.csv）中的数据如下：

```
1,Navin M,98,CS
2,Kavya N,70,CS
3,Nawaz O,80,CS
4,Hari S,80,EC
5,Alex M,50,EC
6,Neenu J,70,EC
7,Bob A,30,EC
8,Anu M,90,AE
9,Sruthi,89,AE
10,Andrew,89,AE
```

按照以下顺序执行脚本：

```
$ ./create_db.sh
Created DB
Created table students

$ ./write_to_db.sh studentdat.csv
Wrote data into DB

$ ./read_db.sh
Department : CS
rank  name  mark
1  Navin M  98
2  Nawaz O  80
3  Kavya N  70

Department : EC
rank  name  mark
1  Hari S  80
2  Neenu J 70
3  Alex M  50
4  Bob A   30

Department : AE
rank  name  mark
1  Anu M    90
2  Sruthi   89
3  Andrew   89
```

9.8.3　工作原理

我们现在来逐个讲解上面的脚本。第一个脚本create_db.sh用来创建数据库students，并在其中创建数据表students。我们需要MySQL的用户名和密码来访问或修改数据库中的内容。变量USER和PASS用来存储用户名和密码。mysql命令用于对MySQL进行操作。该命令可以用-u指定

用户名，用-pPASSWORD指定密码，其他命令参数是数据库名。如果将数据库名作为mysql命令的参数，那么就将使用该数据库，否则就必须用use database_name明确地为SQL查询语句指明使用哪一个数据库进行查询。mysql命令通过标准输入（stdin）接受查询。通过stdin提供多行输入的简便方法是使用<<EOF。出现在<<EOF和EOF之间的文本被作为mysql的标准输入。在CREATE DATABASE语句中，为了避免显示错误信息，我们将stderr重定向到 /dev/null。同样，在创建数据表时，我们也将stderr重定向到 /dev/null，以忽略可能出现的任何错误。然后用退出状态变量$?来检查mysql命令的退出状态，以获知是否已经存在同名的数据库或数据表。如果已经存在，则会显示出一条提示信息；否则，就进行创建。

接下来的脚本write_to_db.sh接受包含学生数据的CSV文件名。我们用while循环读取CSV文件的每一行，因此在每次迭代中都会接收到一行以逗号分隔的数值。然后需要将行内的数值放入SQL查询语句中。要实现这个目的，最简单的方法是用数组存储CSV文件行中的数据项。我们知道数组赋值的形式为array=(val1 val2 val3)，其中**内部字段分隔符**（IFS）是空格。文本行用逗号分隔数值，因此只需要将IFS修改成逗号（IFS=,），就可以轻松地赋值给数组。文本行中以逗号分隔的数据项分别是id、name、mark和department。id和mark是整数，而name和department是字符串（字符串必须进行引用）。name中也可以包含空格。这样一来就和IFS产生了冲突。因此我们应该将name中的空格替换成其他字符（#），在构建查询语句时再替换回来。为了引用字符串，数组中的值要加上 \" 作为前缀和后缀。tr用来将name中的空格替换成 #。最后通过将空格替换成逗号，将 # 替换成空格来构造出查询语句并进行查询。

第三个脚本read_db.sh用来查找各系并打印出每个系的学生排名列表。第一个查询用来找出各系的名称。我们用一个while循环对每个系进行迭代，然后进行查询并按照成绩从高到低的顺序显示学生的详细信息。SET @i:=0是一个SQL构件（SQL construct），用来设置变量i=0。在每一行中，变量i都会增加并作为学生排名来显示。

9.9 用户管理脚本

GNU/Linux是一个多用户操作系统，多个用户可以同时登录并执行多种操作。有一些管理任务会涉及用户管理，这包括为用户设置默认shell、禁用某个账户、禁用某个shell账户、添加新用户、删除用户、设置密码、设置账户有效期等。这则攻略旨在编写一个可以处理此类任务的用户管理工具。

9.9.1 实战演练

让我们看看这个用户管理脚本：

```
#!/bin/bash
#文件名：user_adm.sh
```

```
#用途：用户管理工具

function usage()
{
  echo Usage:
  echo Add a new user
  echo $0 -adduser username password
  echo
  echo Remove an existing user
  echo $0 -deluser username
  echo
  echo Set the default shell for the user
  echo $0 -shell username SHELL_PATH
  echo
  echo Suspend a user account
  echo $0 -disable username
  echo
  echo Enable a suspended user account
  echo $0 -enable username
  echo
  echo Set expiry date for user account
  echo $0 -expiry DATE
  echo
  echo Change password for user account
  echo $0 -passwd username
  echo
  echo Create a new user group
  echo $0 -newgroup groupname
  echo
  echo Remove an existing user group
  echo $0 -delgroup groupname
  echo
  echo Add a user to a group
  echo $0 -addgroup username groupname
  echo
  echo Show details about a user
  echo $0 -details username
  echo
  echo Show usage
  echo $0 -usage
  echo

  exit
}

if [ $UID -ne 0 ];
then
  echo Run $0 as root.
  exit 2
```

```
fi

case $1 in

  -adduser) [ $# -ne 3 ] && usage ; useradd $2 -p $3 -m ;;
  -deluser) [ $# -ne 2 ] && usage ; deluser $2 --remove-all-files;;
  -shell)    [ $# -ne 3 ] && usage ; chsh $2 -s $3 ;;
  -disable) [ $# -ne 2 ] && usage ; usermod -L $2 ;;
  -enable) [ $# -ne 2 ] && usage ; usermod -U $2  ;;
  -expiry) [ $# -ne 3 ] && usage ; chage $2 -E $3 ;;
  -passwd) [ $# -ne 2 ] && usage ; passwd $2 ;;
  -newgroup) [ $# -ne 2 ] && usage ; addgroup $2 ;;
  -delgroup) [ $# -ne 2 ] && usage ; delgroup $2 ;;
  -addgroup) [ $# -ne 3 ] && usage ; addgroup $2 $3 ;;
  -details) [ $# -ne 2 ] && usage ; finger $2 ; chage -l $2 ;;
  -usage) usage ;;
  *) usage ;;
esac
```

输出如下：

```
# ./user_adm.sh -details test
Login: test                Name:
Directory: /home/test                Shell: /bin/sh
Last login Tue Dec 21 00:07 (IST) on pts/1 from localhost
No mail.
No Plan.
Last password change          : Dec 20, 2010
Password expires          : never
Password inactive         : never
Account expires             : Oct 10, 2010
Minimum number of days between password change    : 0
Maximum number of days between password change    : 99999
Number of days of warning before password expires   : 7
```

9.9.2 工作原理

脚本user_adm.sh可以用来执行多种用户管理任务。你可以参考`usage()`中的内容来学习这个脚本的用法。当用户给出的参数不正确或使用-usage选项时，函数`usage()`用来显示脚本不同选项的使用方法。`case`语句用来匹配命令参数，并根据参数执行对应的命令。脚本user_adm.sh合法的命令选项是：`-adduser`，`-deluser`，`-shell`，`-disable`，`-enable`，`expiry`，`-passwd`，`-newgroup`，`-delgroup`，`-addgroup`，`-details`和`-usage`。如果匹配了`*)`分支，那就意味着用户输入了错误的选项，因此要调用`usage()`。对于每一个匹配分支，使用`[ $# -ne 3 ] && usage`。它用来检测参数个数。如果命令参数个数不等于要求的数量，则调用函数`usage()`并退出脚本。要运行用户管理命令，需要以超级用户身份执行脚本。因此要检查用户ID是否为0（root

用户的用户ID是0）。如果用户ID非0，则表明脚本不是以root用户身份执行的,因此显示出要求以root用户身份运行脚本的提示消息并退出。

下面来逐个讲解每个选项。

- -useradd

  useradd命令可以用来创建新用户。命令语法如下：

  useradd USER -p PASSWORD

- 选项-m用来创建home目录。也可以用选项-c FULLNAME提供用户的全名。
- -deluser

  deluser命令用来删除用户。命令语法如下：

  deluser USER

- --remove-all-files用来删除与用户相关的所有文件，包括home目录。
- -shell

  chsh命令用来修改用户的默认shell。命令语法如下：

  chsh USER -s SHELL

- -disable和-enable

  usermod命令用来处理和用户账户相关的若干属性信息。usermod -L USER和usermod -U USER分别用来锁定和解锁用户账户。

- -expiry

  chage命令用来处理用户账户的过期信息。命令语法如下：

  chage -E DATE

  其他选项如下：

  - -m MIN_DAYS（将更改密码的最小天数修改成MIN_DAYS）；
  - -M MAX_DAYS（设置密码有效的最大天数）；
  - -W -WARN_DAYS（设置在前几天提醒需要更改密码）。

- -passwd

  passwd命令用来更改用户密码。命令语法如下：

  passwd USER

命令会提示输入新的密码。

- -newgroup和addgroup

  addgroup命令会为系统添加一个新的用户组。命令语法如下：

```
addgroup GROUP
```

  要将已有的用户添加到一个组，可以使用：

```
addgroup USER GROUP
-delgroup
```

  delgroup命令会删除一个用户组。命令语法如下：

```
delgroup GROUP
```

- -details

  figer USER命令会显示用户信息，这包括用户的home目录、上一次登录的时间、默认shell等。chage -l命令会显示用户账户的过期信息。

9.10 图像文件的缩放及格式转换

我们大家都会使用数码相机，也会从网上获取数码照片。如果需要处理大量图像文件，我们可以轻松地用脚本进行批量处理。这通常会涉及到的任务是调整照片的大小，还有转换图像格式（例如，将JPEG格式转换成PNG格式）。当从数码相机中取到照片时，大分辨率的图片体积都比较大，我们可能需要减少图片的大小，以便于存储并通过电子邮件发送。因此就需要调整图片的大小来降低分辨率。这则攻略将讨论如何用脚本管理图片。

9.10.1 预备知识

我们要用到convert命令，它来自Imagemagick软件包，该软件包中包含了大量出色的图像处理工具，拥有各类选项的丰富，能够处理多种图像格式。大多数GNU/Linux发布版中并没有预装Imagemagick。你得自己手动安装。更多的信息请访问www.imagemagick.org。

9.10.2 实战演练

将一种图像格式转换为另一种图像格式：

```
$ convert INPUT_FILE OUTPUT_FILE
```

例如：

```
$ convert file1.png file2.png
```

我们可以通过指定缩放比或输出图像的宽度和高度来调整图像。

指定WIDTH（宽度）或HEIGHT（高度）来缩放图像：

```
$ convert image.png -resize WIDTHxHEIGHT image.png
```

例如：

```
$ convert image.png -resize 1024x768 image.png
```

必须提供WIDTH或HEIGHT，这样才能够自动计算其他数值，以便在保留图像比例的同时进行缩放。

```
$ convert image.png -resize WIDTHx image.png
```

例如：

```
$ convert image.png -resize 1024x image.png
```

指定百分比缩放图像：

```
$ convert image.png -resize "50%" image.png
```

让我们看一个用于图像管理的脚本：

```
#!/bin/bash
#文件名：image_help.sh
#用途:图像管理脚本

if [ $# -ne 4 -a $# -ne 6 -a $# -ne 8 ];
then
  echo Incorrect number of arguments
  exit 2
fi

while [ $# -ne 0 ];
do

  case $1 in
  -source) shift; source_dir=$1 ; shift ;;
  -scale) shift; scale=$1 ; shift ;;
  -percent) shift; percent=$1 ; shift ;;
  -dest) shift ; dest_dir=$1 ; shift ;;
  -ext) shift ; ext=$1 ; shift ;;
  *) echo Wrong parameters; exit 2 ;;
  esac;

done
```

```
for img in `echo $source_dir/*` ;
do
  source_file=$img
  if [[ -n $ext ]];
  then
    dest_file=${img%.*}.$ext
  else
    dest_file=$img
  fi

  if [[ -n $dest_dir ]];
  then
    dest_file=${dest_file##*/}
    dest_file="$dest_dir/$dest_file"
  fi

  if [[ -n $scale ]];
  then
    PARAM="-resize $scale"
  elif [[ -n $percent ]];    then
    PARAM="-resize $percent%"
  fi

  echo Processing file : $source_file
  convert $source_file $PARAM $dest_file

done
```

下面是输出样例，将目录sample_dir中的图像调整到原来的20%：

```
$ ./image_help.sh -source sample_dir -percent 20%
Processing file :sample/IMG_4455.JPG
Processing file :sample/IMG_4456.JPG
Processing file :sample/IMG_4457.JPG
Processing file :sample/IMG_4458.JPG
```

将图像宽度调整到1024像素：

```
$ ./image_help.sh -source sample_dir -scale 1024x
```

把`-ext png`加入上面的命令，使文件格式转换成PNG。

将文件缩放或转换到指定的目录：

```
$ ./image_help.sh -source sample -scale 50% -ext png -dest newdir
# newdir作为目的目录
```

9.10.3　工作原理

上面的脚本image_help.sh可以接受多个命令行参数，例如-source、-percent、-scale、-ext和-dest等。每个选项的简短描述如下。

- -source用于指定图像源目录。
- -percent用于指定缩放比例，-scale用于指定缩放宽度与高度。
- -percent与-scale不能同时出现，只能使用其中之一。
- -ext用于指定目标文件格式。-ext是一个可选的选项。如果没有指定，那么不执行格式转换。
- -dest为缩放或转换格式后的文件指定目标目录。该选项也是可选的。如果没有指定，目标目录则和源目录相同。脚本的第一步就是检查命令行参数的数量是否正确，可以出现的参数数量分别是4、6或8。

借助while循环和case语句，我们将命令行参数解析到对应的变量。$#是一个特殊的变量，它可以返回参数的数量。shift命令每执行一次，就将命令行参数向左移动一个位置，这样我们就不需要使用变量$1、$2、$3等，而只用一个$1就可以对命令参数逐个访问了。case语句用来匹配$1的值，就像C语言中的switch语句一样。如果匹配了某个case分支，就执行对应的语句。每一个case分支都以;;作为结尾。一旦将所有的参数都解析到变量percent、scale、source_dir、ext和dest_dir中，就用for循环对源目录中的每一个文件进行迭代，并执行对应的转换操作。

如果变量ext已定义（也就是说-ext作为命令参数出现），就将目标文件的扩展名从source_file.extension更改为source_file.$ext。接下来检查是否提供了-dest选项。如果指定了目标目录，则使用文件名切片将源路径中的目录替换成目标目录，从而形成目标文件路径。随后构造出convert命令的参数，用以执行缩放（-resize widthx 或-resize perc%）。参数构造完毕之后，用对应的参数执行convert命令。

9.10.4　参考

2.12节讲解了如何提取部分文件名。

9.11　从终端截图

截图是另一种常见的计算机用户日常操作。对于维护GUI应用以及自动化管理的系统管理员而言，这种操作更为重要。当特定事件出现时，关键就是要抓取截图，以便知道GUI应用到底出了什么事。

9.11.1 预备知识

我们要使用一个来自ImageMagick软件包的工具，该工具包在上一个攻略中已经有所涉及。请使用软件包管理器自行安装。

9.11.2 实战演练

下面我们使用ImageMagick中的`import`工具进行截图。

(1) 取整个屏幕：

```
$ import -window root screenshot.png
```

(2) 手动选择部分区域进行抓取：

```
$ import screenshot.png
```

(3) 抓取特定窗口：

```
$ import -window window_id screenshot.png
```

使用命令`xwininfo`，点击需要抓取截图的窗口来获得`window_id`。然后将`window_id`传递给`import`命令的`-window`选项。

9.12 管理多个终端

如果你频繁地使用shell，大概会注意到有时候需要同时访问多个终端。如果你使用的是如Konsole这类图形化终端模拟器，就得借助多个标签（tab）才能满足这种需求。

但是如果没有图形化终端模拟器该怎么办呢？或者说登录到远程主机，希望使用多个shell，又该怎么办呢？在后一种情况中，打开多个`ssh`连接基本上只能浪费网络带宽，拖慢运行速度而已。接下来我们会看到如何在避免这些问题的同时获得多个shell。

9.12.1 预备知识

要实现这一点，需要使用一个叫做GNU screen的工具。如果你使用的发布版中没有预装screen，请使用软件包管理器自行安装。

9.12.2 实战演练

- 创建新的screen窗口：从shell中运行screen来创建一个新的screen。你会看到一条欢迎消

息以及一些该软件的相关信息。按空格键或回车键开始。接着会获得一个新的shell用以输入命令。要创建新窗口（基本上相当于一个新shell），使用Ctrl+A+C（区分大小写）。

- **查看已打开的窗口列表**：在运行screen时，如果要查看所有打开窗口，使用Ctrl+A+"。
- **在窗口之间切换**：我们通常需要以下一个/前一个（next/previous）的方式在打开的窗口之间进行切换。可以使用Ctrl+A和Ctrl+N切换到下一个窗口，使用Ctrl+A和Ctrl+P切换到前一个窗口。
- **关联与脱离screen**：screen具备一个有用的特性，能够让你保持、载入screen会话，用screen的术语来说，叫做脱离（detaching）与关联（attaching）。使用Ctrl+A和Ctrl+D脱离当前screen会话。要关联到一个已有的screen会话，可以使用：

```
screen -r -d
```

该命令告诉screen关联到上一个screen会话。如果已脱离的会话不止一个，screen会用列表输出会话，然后可以使用下述命令：

```
screen -r -d PID
```

这里，`PID`是你想关联到的screen会话的PID。

Linux/GNU是一款非凡的操作系统，拥有稳定可靠且极其强大的开发环境。作为与操作系统进行沟通的原生界面，shell能够控制整个操作系统的运作，是与Linux进行交互最灵活的手段。本书向读者展现了如何有效地利用shell完成复杂的任务。从shell的基础知识开始，学习简单命令的用法，对各类文件进行操作。随后讲解了文本处理、Web交互、备份、监视以及其他系统管理任务。第2版进行了全面修订，精选极具实用价值的技巧，让你的日常工作更加轻松。

- 考察了各类日常任务以及如何利用shell命令更快速地解决问题。
- 编写脚本从Web中挖掘数据，然后利用几行代码进行处理。
- 综合利用多种工具来解决问题。
- 在脚本中与简单的Web API进行交互。
- 任务的执行及自动化，例如利用归档工具实现自动备份和恢复。
- 创建及维护文件和文件夹归档，利用shell进行压缩和加密。
- 利用shell脚本设置以太网和无线局域网。
- 利用日志监视网络活动。

[PACKT] PUBLISHING

图灵社区：iTuring.cn

热线：(010)51095186 转 600

分类建议 计算机 / 程序设计 / Linux

人民邮电出版社网址：www.ptpress.com.cn

ISBN 978-7-115-33921-8

定价:59.00元